

Aus dem Institut für Medizinische Statistik und Dokumentation  
der Universität Lahn-Gießen  
Leiter: Prof. Dr. J. Dudeck

Anwendbarkeit  
von Datenorganisationsprinzipien  
auf kleineren EDV-Anlagen im Bereich der  
medizinischen Dokumentation und Statistik

Inaugural-Dissertation  
zur Erlangung des Doktorgrades  
der Humanbiologie der Philipps-Universität Marburg  
Fachbereich Humanmedizin

vorgelegt von  
Hans-Jürgen Friedrich  
aus Aussig

Marburg 1977



Aus dem Institut für Medizinische Statistik und Dokumentation  
der Universität Lahn-Gießen  
Leiter: Prof. Dr. J. Dudeck

**Anwendbarkeit  
von Datenorganisationsprinzipien  
auf kleineren EDV-Anlagen im Bereich der  
medizinischen Dokumentation und Statistik**



Inaugural-Dissertation  
zur Erlangung des Doktorgrades  
der Humanbiologie der Philipps-Universität Marburg  
Fachbereich Humanmedizin

vorgelegt von  
Hans-Jürgen Friedrich  
aus Aussig

Marburg 1977

Angenommen vom Fachbereich Humanmedizin  
der Universität Marburg am



Gedruckt mit der Genehmigung des Fachbereichs

Dekan: Prof. Dr. NEURATH

Referent: Prof. Dr. IHM

Correferent: Prof. Dr. DUDECK

Tag der mündlichen Prüfung 21. 12. 1977

|        |                                                                       |    |
|--------|-----------------------------------------------------------------------|----|
| Teil I | <u>Darstellung von Daten(bank)organisations-</u><br><u>prinzipien</u> | 1  |
| 1      | <u>Einleitung</u>                                                     | 1  |
| 1.1    | Gegenstand und Ziel der Arbeit                                        | 1  |
| 1.2    | Terminologischer Überblick                                            | 2  |
| 2      | <u>Elemente der Daten(bank)organisation</u>                           | 9  |
| 2.1    | Elementare Organisationseinheiten                                     | 9  |
| 2.2    | Satzbegriff                                                           | 10 |
| 2.3    | Darstellung von Daten durch Schlüssel                                 | 16 |
| 3      | <u>Methoden der Daten(bank)organisation</u>                           | 20 |
| 3.1    | Generelle Speicherungsmöglichkeiten                                   | 20 |
| 3.1.1  | Fortlaufende Speicherung                                              | 21 |
| 3.1.2  | Gestreute Speicherung                                                 | 24 |
| 3.1.3  | Kombination von fortlaufender und gestreuter<br>Speicherung           | 27 |
| 3.2    | Such- und Änderungsverfahren mit linearen<br>Listen                   | 28 |
| 3.2.1  | Suchverfahren ohne Zeiger                                             | 30 |
|        | a) Suchverfahren ohne Adressrechnung                                  |    |
|        | (1) Sequentielle Suche                                                |    |
|        | (2) m-Wege-Suche                                                      |    |
|        | (3) binäre (duale) Suche                                              |    |
|        | b) Suchverfahren mit Adressrechnung                                   |    |
|        | (1) Direkte Adressierung                                              |    |
|        | (2) Indirekte Adressierung (HASH)                                     |    |
| 3.2.2  | Suchverfahren mit Zeigern                                             | 44 |
|        | a) Adressverkettung                                                   |    |
|        | (1) Einfache Verkettung                                               |    |
|        | (2) Doppelte Verkettung                                               |    |
|        | (3) Feldwertverkettung                                                |    |
|        | (4) Mehrfachverkettung                                                |    |
|        | (5) Hierarchische Verkettung                                          |    |

|       |                                                                                                                          |     |
|-------|--------------------------------------------------------------------------------------------------------------------------|-----|
| b)    | Indizierung                                                                                                              |     |
| (1)   | Primärindizierung                                                                                                        |     |
| (2)   | Sekundärindizierung (Invertierung)                                                                                       |     |
| (3)   | Mehrdimensionale Indizierung                                                                                             |     |
| (4)   | Binärmatrizen                                                                                                            |     |
| 3.3   | Such- und Änderungsverfahren mit Bäumen<br>(nicht-lineare Listen)                                                        | 54  |
| 3.3.1 | Binärbäume                                                                                                               | 56  |
| a)    | Verarbeitungsformen an Binärbäumen                                                                                       |     |
| b)    | AVL-Bäume                                                                                                                |     |
| c)    | Binärbaumdarstellung von Mehrwegbäumen                                                                                   |     |
| 3.3.2 | Mehrwegbäume                                                                                                             | 64  |
| a)    | Darstellung von Mehrwegbäumen mit Index-<br>knoten                                                                       |     |
| b)    | B-Bäume                                                                                                                  |     |
| 4     | <u>Umfassende Datenbankmodelle</u>                                                                                       | 73  |
| 4.1   | CODASYL                                                                                                                  | 73  |
| 4.1.1 | Bericht der CODASYL-Data-Base-Task-Group                                                                                 | 73  |
| 4.1.2 | Datenbasis-Verwaltungssystem                                                                                             | 76  |
| 4.1.3 | Datenstrukturen                                                                                                          | 79  |
| 4.1.4 | Schema und Unterschema                                                                                                   | 83  |
| 4.1.5 | Sammlungskonzept                                                                                                         | 84  |
| 4.1.6 | Erweiterung des Sammlungskonzepts und Schwierigkeiten bei der Anwendung auf komplexe Datenstrukturen                     | 89  |
| 4.2   | Relationalmodell                                                                                                         | 94  |
| 4.2.1 | Begriffe                                                                                                                 | 95  |
| 4.2.2 | Normalisierung                                                                                                           | 98  |
| 4.2.3 | Operationen mit Tupeln                                                                                                   | 102 |
| 4.2.4 | Untersuchung der Implementierungsmöglichkeiten des "join"-Operators in einer gebräuchlichen Programmiersprache (FORTRAN) | 105 |
| a)    | Bilden des Cartesischen Produkts                                                                                         |     |
| b)    | Zusammenführen von sortierten Files                                                                                      |     |
| c)    | Verbinden von Indizes                                                                                                    |     |

|         |                                                                                                                                    |     |
|---------|------------------------------------------------------------------------------------------------------------------------------------|-----|
| Teil II | <u>Praktische Anwendungen von Daten(bank)organisationsprinzipien</u>                                                               | 110 |
| 1       | <u>Thesen</u>                                                                                                                      | 110 |
| 2       | <u>Überlegungen zur Auswahl einer Beschreibungssprache unter Berücksichtigung ihrer Übertragbarkeit in FORTRAN</u>                 | 113 |
| 2.1     | Regeln zur Übertragung von Algorithmen aus LISP in FORTRAN                                                                         | 114 |
| 2.2     | Umsetzung von Strukturblöcken in FORTRAN                                                                                           | 116 |
| 3       | <u>Interaktion mit einer Datenbank</u>                                                                                             | 121 |
| 3.1     | Eingabe von Daten in die Datenbank (Formatgenerator; s. Anhang: DEP)                                                               | 121 |
| 3.2     | Abfrage von Daten (Formelinterpreter; s. Anhang: CONTRA)                                                                           | 123 |
| 4       | <u>Darstellung von drei Systemen als Beispiele zur Realisierung von Datenbankorganisationsprinzipien auf kleineren EDV-Anlagen</u> | 130 |
| 4.1     | Recherche-Moduln zur Erschließung medizinischer Daten (s. Anhang: REMED)                                                           | 130 |
| 4.1.1   | Mögliche Datenstrukturen in REMED                                                                                                  | 130 |
|         | a) Usual File                                                                                                                      |     |
|         | b) Inverted File                                                                                                                   |     |
| 4.1.2   | Abfragesprache                                                                                                                     | 133 |
|         | a) Stringzerlegung                                                                                                                 |     |
|         | b) Lexikalische Analyse                                                                                                            |     |
|         | c) Bilden eines polnischen Strings                                                                                                 |     |
|         | d) Überführen in eine Befehlsfolge                                                                                                 |     |
| 4.1.3   | Beispiele für die Interaktion mit der Datenbank                                                                                    | 140 |
|         | a) Formulierung eines Auswahlausdrucks                                                                                             |     |
|         | b) Formulierung von Suchfragen                                                                                                     |     |
|         | c) Anforderung von Output-Listen                                                                                                   |     |
| 4.2     | Interaktives Auswertungs-System für medizinisch-statistische Daten (IAS)                                                           | 142 |

|          |                                                                                                              |     |
|----------|--------------------------------------------------------------------------------------------------------------|-----|
| 4.2.1    | Allgemeines                                                                                                  | 142 |
| 4.2.2    | Datenstrukturen von medizinisch-statistischen Daten                                                          | 145 |
|          | a) Überlegungen zur Strukturänderung                                                                         |     |
|          | b) Überlegungen zur Darstellungsänderung                                                                     |     |
|          | c) Realisierung von Struktur- und Darstellungsänderung durch Programme (s. Anhang TUP, CHANGE)               |     |
|          | (1) Darstellungsänderung                                                                                     |     |
|          | (2) Strukturänderung                                                                                         |     |
| 4.2.3    | Trennung zwischen Datenstruktur und Speicherstruktur und ihre Realisierung durch Programme (s. Anhang: DZP)  | 153 |
|          | a) Realisierung bei festgelegter Relation zwischen logischem und physikalischem Satz                         |     |
|          | b) Realisierung bei nicht festgelegter Relation zwischen logischem und physikalischem Satz                   |     |
| 4.3      | Retrievalsystem zur ernährungswissenschaftlichen Literaturdokumentation (RS-II)                              | 155 |
| 4.3.1    | Entwicklung einer Methode zur Invertierung größerer Datenbestände auf kleineren EDV-anlagen (s. Anhang: MBQ) | 155 |
| 4.3.2    | Möglichkeiten zur Realisierung von Mengenoperationen (s. Anhang: SETOP)                                      | 159 |
| Teil III | <u>Diskussion und Zusammenfassung</u>                                                                        | 160 |
| 1.       | <u>Diskussion und Schlußfolgerungen</u>                                                                      | 160 |
| 2.       | <u>Zusammenfassung</u>                                                                                       | 166 |
|          | Literaturverzeichnis                                                                                         | 169 |
|          | Anhang                                                                                                       | 178 |

# I Darstellung von Datenbankorganisationsprinzipien =====

## 1 Einleitung

### 1.1 Gegenstand und Ziel der Arbeit

Das Bild der Datenverarbeitung, auch in der Medizin, hat sich in den letzten Jahren erheblich geändert. Zum einen ist eine gewisse Ernüchterung in der Beurteilung des Einsatzes und des Nutzens der EDV eingetreten; wir befinden uns gegenwärtig in einer Phase der kritischen Bestandsaufnahme und des Neuüberdenkens. Als erster äußerlich sichtbarer Ausdruck hierfür mag das Thema der diesjährigen Jahrestagung der Gesellschaft für medizinische Dokumentation, Informatik und Statistik (1977) gelten:

"Informationsverarbeitung in der Medizin - Wege und Irrwege".

Zum anderen geht die technologische Entwicklung zum Kleinrechner (Mikroprozessor) und zu einer damit verbundenen Dezentralisierung, die ihrerseits wiederum neue Organisationsformen, wie Netzwerke, nach sich zieht.

Die oben angesprochene Ernüchterung ist u.E. durch die Erfahrung einer "Überdimensionierung" entstanden; sie weicht heute einer geforderten "Angemessenheit". Der Begriff der Überdimensionierung soll in zweierlei Hinsicht verstanden werden: Überkapazität bei den maschinellen Ressourcen auf der einen Seite und Anspruch auf Totallösungen auf der anderen. Entsprechend bezieht sich der Begriff der Angemessenheit ebenso auf eine angepaßte Hardwareausstattung, wie auf den Verzicht, Gesamtlösungen erarbeiten zu wollen.

Will man mit kleineren EDV-Anlagen eine nutzbringende medizinische Datenverarbeitung im Bereich der Dokumentation und Statistik durchführen, so muß vor allem auch die Frage der Bewältigung von größeren Datenmengen gelöst werden. Auf bestehende Datenbanksysteme kann dabei in der Regel

nicht zurückgegriffen werden, da diese kaum universell verwendbar sind für verschiedene Anlagen und für die verschiedenen Problembereiche.

Ziel der vorliegenden Arbeit ist es, erstmals systematisch zu untersuchen, welche der allgemeinen Datenbankorganisationsprinzipien auf kleineren EDV-Anlagen anwendbar sind, wie diese in einer kompatiblen höheren Programmiersprache realisiert werden können, und wie sie, in Moduln zu einem Tool-Pool zusammengeschlossen, zum Aufbau von Systemen verwendet werden können, die ihrerseits auf spezifische Probleme im medizinischen Bereich zugeschnitten sind und diese "angemessen" lösen können.

Am Beispiel eigener umfangreicher Systementwicklungen sollen Lösungsmöglichkeiten der genannten Probleme aufgezeigt werden.

## 1.2 Terminologischer Überblick

Die Terminologie im Bereich "Datenbank" ist vielfältig und uneinheitlich. Zu einer Begriffsbestimmung benötigen wir neben der inhaltlichen Komponente ("bestimmte Art der Sammlung von Daten") die funktionale Komponente ("Organisationsprinzipien zur Speicherung und Wiedergewinnung von Daten").

Zur inhaltlichen Komponente: Daten werden in einer Datenbasis oder in einer Datei gespeichert. Dateien bestehen aus einer Anzahl von Sätzen, die jeweils eine Reihe von Feldern enthalten, deren Inhalte die zu speichernden Daten sind. Jedes einzelne dieser Felder hat eine besondere Bedeutung, da sein Inhalt die Reihenfolge, in der die Sätze gespeichert werden, bestimmt. Ein Feld enthält einen Ordnungsbegriff der Datei (Index, Hauptschlüssel oder Sortierbegriff).

Zur funktionalen Komponente: Eine Anordnung von Daten zwecks Wiederfindung zur Verarbeitung nennt man Datenorganisation. Sie ist immer dann notwendig, wenn die Daten gespeichert werden müssen, um zu einem späteren Zeitpunkt verarbeitet zu werden. Datenorganisation besteht aus Verfahren oder Methoden, mit denen die Speicherung der Daten erfolgt und ihre spätere Wiedergewinnung ermöglicht wird.

WEDEKIND definiert den Begriff Datenbank in Abhängigkeit von der Datenorganisation: "Wenn die Datenorganisation einen gewissen Komplexitätsgrad erreicht, spricht man von einer Datenbank" (WEDEKIND, 1970).

Im folgenden wird zunächst die Datensammlung ohne explizite Datenbankorganisation (d.h. das bisherige Vorgehen) dargestellt.

Ausgehend von einem algorithmisierten, bzw. programmierten Problem wird geprüft, welche Daten im Ablauf eines Programms gebraucht werden. Dementsprechend werden die Daten, zugeschnitten auf das spezielle Programm, in einer Datei organisiert. Bei der Lösung eines größeren Problems sind mehrere Programme notwendig, wobei für jedes dieser Programme dann wieder eine spezielle Form der Datei benötigt wird. Man bezeichnet dieses Vorgehen als programmorientiert. Auf der anderen Seite muß jedes Programm auch Angaben darüber enthalten, welche Dateien benötigt werden, wie diese Dateien aufgebaut sind, welche Daten aus den Dateien benötigt werden und wie auf diese Daten zuzugreifen ist.

Diese eben beschriebene Form der Datensammlung enthält eine Reihe von Nachteilen, die im folgenden aufgezählt werden sollen (MERTEN, 1974):

- Da mit der Zeit eine Vielzahl von Dateien entsteht, wird der Überblick über die bestehenden Dateien und deren Benutzung erschwert.

- Die große Zahl der Dateien führt dazu, daß viele Daten in mehreren Dateien, also redundant abgespeichert werden. Bei Änderungen in solchen Dateien müssen entsprechend in allen anderen betroffenen Dateien ebenfalls Änderungen durchgeführt werden, was eine erhöhte Fehleranfälligkeit zur Folge hat.
- Da die meisten Dateien für ein ganz bestimmtes Verfahren erstellt wurden, müssen sie umorganisiert werden, wenn ein neues anderes Verfahren auf dieselben Daten zugreifen soll, d.h. zur Durchführung anderer Verarbeitungen mit denselben Daten müssen die Dateien umstrukturiert werden, oder es müssen neue Dateien aufgebaut werden, die dieselben Daten enthalten.
- Da die meisten Dateien zudem nur nach einem Ordnungsprinzip strukturiert sind, bei vielen Problemen diese aber nach einem anderen Ordnungsbegriff verarbeitet werden sollen (triviales Beispiel: statt nach dem Namen soll nach dem Geburtstag sortiert werden), sind häufig zeitraubende Prozesse notwendig.
- Ein Nachteil, der sich aus den Konfigurationen der EDV-Anlagen ergibt, ist die Tatsache, daß für aufwendige Programme meist sehr viele Eingabedateien nötig sind, die Anzahl der logischen bzw. physikalischen Eingabegeräte aber beschränkt ist. Vor der Verarbeitung von Dateien müssen diese also zuerst zu einer Datei zusammen gemischt bzw. vereinigt werden, wodurch zusätzlich viel Zeit benötigt wird.

Aus den genannten Nachteilen lassen sich folgende Forderungen an eine Datenbankorganisation (d.h. für das heutige Vorgehen) ableiten (LUTZ & KLIMESCH, 1971).

## 1. Zusammenfassung mehrerer Dateien

Ein Ziel hierbei ist die weitgehende Beseitigung der Redundanz in der Datenspeicherung. Zusammenfassungen mehrerer Dateien ermöglichen die Zusammenfassung logisch zusammengehöriger Dateien und damit eine Integration der gespeicherten Daten.

## 2. Strikte Trennung von Daten und Programmen

Werden Dateien unabhängig von den Verarbeitungsprogrammen aufgebaut und in der Datenbasis gespeichert, geschieht eine konsequente Trennung von Datenverarbeitung und Datenverwaltung. Es werden nicht mehr zuerst Programme geschrieben, dann die dafür benötigten Daten organisiert, sondern es wird umgekehrt vorgegangen, indem zuerst Dateien organisiert und aufgebaut werden und erst dann Programme folgen, die von diesen Dateien unabhängig entwickelt sind, um die Daten zu benutzen (Datenunabhängigkeit). Eingabe, Speicherung und Änderung der Daten erfolgt nicht mehr im Zusammenspiel zwischen Programmen und Dateien, sondern über einen Mittler, über ein Datenverwaltungssystem. Die Trennung von Programmen und Daten bewirkt unter anderem, daß die Änderung der Programme nicht eine Änderung von Dateien nach sich zieht.

Einen Schritt in Richtung Datenunabhängigkeit für den Problemkreis "statistische Auswerteprogramme" stellen die unten beschriebenen Module des Interaktiven Auswertesystems (IAS) dar, das im Zusammenhang mit einer generalisierten Schnittstelle zu sehen ist.

3. Zugriff zu den Daten nicht über die Programme, sondern über ein besonderes Datenzugriffssystem

Da der Zugriff zu den Daten über ein eigenes Datenzugriffssystem erfolgt, das zur Datenbank gehört und die Schnittstelle zwischen Programmen und Datenbanken bildet, werden alle für die Programme benötigten Daten einheitlich über dieses Datenzugriffssystem beschafft, so daß der Endbenutzer von der Organisation der Datenbeschaffung entlastet wird und sich auf die Definition der benötigten Daten beschränken kann. Es entfallen Misch- und Sortiervorgänge, die beim Ablauf mehrerer Programme bei herkömmlichen Verfahren notwendig sind.

4. Möglichkeit, aus den gespeicherten Daten beliebige Auswertungen zu gewinnen

(Wir werden später sehen, daß es sich hier um das Problem der generalisierten Schnittstelle handelt.)

Die von den Programmen unabhängige Speicherung der Daten in der Datenbank beinhaltet die Möglichkeit, beliebige Auswertungen aus den gespeicherten Daten vorzunehmen. Da die Daten nicht mehr für ein einzelnes Programm organisiert werden, muß jedoch garantiert sein, daß die Programme trotzdem die benötigten Daten erhalten. Die Forderung lautet zwar nicht, daß Speicherdaten für jeden beliebigen Zweck zur Verfügung gestellt werden müssen ("beliebiger Zweck" im Sinne von "verarbeitungsbezogener Zur-Verfügung-Stellung"), dennoch sollte es für alle Fragen, die nach ihrer Art bereits vorklassifiziert sind, auf einfachste Art möglich sein, Daten aus Dateien umzuformen, um diese zu aktivieren und als Information den Programmen zur Verfügung zu stellen.

## 5. Mehrfachbenutzung

Wenn Programme und Dateien getrennt werden, so muß bei gleichzeitiger Mehrfachbenutzung derselben Dateien allerdings auch gewährleistet sein, daß andere Programme den Zugriff zur Datenbank nicht behindern.

Eine Arbeitsdefinition für Datenbank (Datenbanksystem) könnte nach dem oben Gesagten wie folgt lauten: Eine Datenbank ist ein Organisationsprinzip zur Sammlung von Datenbeständen in Dateien oder Datenbasen, die durch ein Datenverwaltungssystem verwaltet werden und über ein Datenzugriffssystem (mehreren Benutzern gleichzeitig) für beliebige Auswertungen zur Verfügung stehen.

Von Benutzer im Sinne der obigen Definition kann man sprechen, wenn ein Programm vorliegt, das Daten aus dem Datenbanksystem abrufen, oder wenn eine direkte Kommunikation mit der Datenbank von einem menschlichen Benutzer vorliegt. Im Fall des Zugriffs von Programmen auf die Datenbank taucht das Problem der Schnittstellendefinition auf, im Fall des Menschen als Benutzer entsteht die Problematik der Abfragesprachen und der Implementierung von Dialog-Abfragesprachen.

Zwar haben beide Benutzerklassen ihre spezifischen Probleme, doch sollte an leistungsfähige Datenbanksysteme die Forderung gestellt werden, beide Kommunikationsformen zu ermöglichen, insbesondere da die Daten, die benötigt werden, fast immer die gleichen sind.

Die Realisierungen von Abfragesprachen bzw. Programmen, die auf die Daten über ein Datenzugriffssystem zugreifen, nennt man Abfragesprache bzw. Abfragemodul, wobei die Ergebnisse der Operationen Informationen in be-

stimmter Form sind, die ihrerseits mit weiteren Programmen verarbeitet werden können bzw. direkt zu Entscheidungen herangezogen werden können.

Handelt es sich um Abfragemodule, so findet man häufig die Erweiterung des Abfragemoduls durch Methoden und Modellbanken. Auf die Realisierung von Abfrage- oder Recherchemodulen wird später eingegangen (Recherche-Modul für eine EKG-Datenbank).

Der Begriff Datenbanksystem ist schließlich noch abzugrenzen gegenüber dem Begriff Dokumentationssystem. Als wichtigstes Unterscheidungskriterium kann hierbei die Unterscheidung zwischen formatierten Daten und unformatierten Daten (strukturiert vs. unstrukturiert, numerisch vs. nicht-numerisch) herangezogen werden. Während bei formatierten Daten durch den Aufbau der Sätze und Felder eine feste Struktur für die zu speichernden Daten vorgegeben ist bzw. sich bei variablen Satzlengthen die Bedeutung der Elemente generieren läßt, liegen bei unformatierten Daten nur Informationen über die Daten selbst vor. Die Bedeutung der Sätze und Wörter ist kontextabhängig (zum Gesamtdokument bzw. zu den übrigen Dokumenten).

Im allgemeinen wird bei formatierten Daten von Datenbanken und Datenbanksystemen gesprochen, bei unformatierten Daten dagegen von Dokumentationssystemen. In den angelsächsischen Ländern ist der Begriff "information storage and retrieval" oder "information retrieval" als Oberbegriff für beide Systeme gebräuchlich. Für ihn fehlt allerdings eine adäquate deutsche Übersetzung.

## 2 Elemente der Daten(bank)organisation

### 2.1 Elementare Organisationseinheiten

Im Zusammenhang mit den elementaren Datenanordnungen sollen zunächst einige in der Literatur gebräuchliche Begriffe vorangestellt werden, wobei betont werden muß, daß die Terminologie in diesem Gebiet sehr uneinheitlich gehandhabt wird.

Die unterste Datenebene ist das Bit, mehrere Bits (im allgemeinen 8) ergeben ein Byte und mehrere Bytes ein Feld. Hier beginnt bereits die terminologische Mehrdeutigkeit, da man in dem letzten Fall auch von Wort sprechen kann. Mehrere Worte zusammen ergeben ein Segment; auch dieser Begriff wird in der Literatur unterschiedlich gehandhabt. Schließlich gibt es Zusammenfassungen von mehreren Segmenten zu einem Satz. Die Satzzusammenfassung kann man Datei nennen, und mehrere Dateien können schließlich eine Datenbank bilden.

Die organisatorischen Einheiten außer Bit und Byte können alle nach Namen und Inhalt unterschieden werden. Man spricht daher von Feldnamen und Feldinhalten bzw. Feldwerten. Statt der Begriffe Feldname und Feldinhalt lassen sich auch die Begriffe Merkmal und Merkmalsausprägung verwenden.

Die Verwendung dieser letztgenannten Terminologie, Merkmal und Merkmalsausprägung, ist dann sinnvoll, wenn ausgedrückt werden soll, daß sich die Daten eines Satzes immer auf ein bestimmtes Objekt (Person, Sache oder Begriff) beziehen. Dieses Objekt ist dann Merkmalsträger (BAGINSKI, 1970) und wird durch den Ordnungsbegriff identifiziert. So kann z.B. eine Personalnummer als Ordnungsbegriff fungieren und in jedem Satz einen Patienten als

Merkmalsträger identifizieren. Die übrigen Felder des Satzes können Merkmalsausprägungen enthalten, die diesen Patienten beschreiben. Jede dieser Merkmalsausprägungen läßt sich einem Merkmal zuordnen, das durch den Feldnamen bezeichnet wird. Ein Feld kann grundsätzlich variable oder feste Länge haben.

Der bei der Verwendung von Feldern variabler Länge eingesparte Speicherplatz wird allerdings erkaufte durch einen erhöhten Rechenaufwand, da vor der Verarbeitung des Feldes immer erst seine Länge festgestellt werden muß (HARTMAN, MATTHES & PROEME, 1968). Sobald aber variable Feldlängen zugelassen sind, impliziert das auch, daß die Segmente und Sätze unterschiedlicher Länge sein können. Es treten dann gegenüber Sätzen fester Länge zusätzliche organisatorische Probleme bei der Speicherung wiedergewählter Sätze auf. Sätze können auch eine variable Länge aufweisen, wenn einzelne Segmente oder Felder in einem Satz mehrfach auftreten, z.B. bei Mehrfachdiagnosen, deren Anzahl nicht beschränkt wird pro Patient. Hier unterscheiden sich die Sätze auf jeden Fall in ihrer Länge, auch wenn die einzelnen Segmente eine feste Länge haben, da die Häufigkeit des Auftretens je Satz unterschiedlich ist. Man spricht hier von Vielfach- oder Mehrfachsegmenten bzw. repeating groups.

Auf der hierarchischen Ebene der Datei gibt es fast immer eine variable Länge (Umfang), da die Anzahl der Sätze einer Datei nur in Ausnahmefällen konstant ist. Für die Datenbank selbst gilt grundsätzlich, daß sie in ihrem Umfang variabel ist.

## 2.2 Satzbegriff

Von den oben kurz behandelten Begriffen hat der Satz in der Datenorganisation die größte Bedeutung, und zwar

deswegen, weil die Speicherung der Daten vom Aufbau der Sätze abhängt und weil die Daten als Sätze wiedergewonnen werden. Der Satz enthält die kleineren Elemente Segment und Feld und damit die wesentlichen datenorganisatorischen Bestandteile. Die Bedeutung der Datei ist demgegenüber untergeordnet, da eine Datei grundsätzlich als eine Sammlung oder eine Wiederholung von Sätzen angesehen werden kann, deren Struktur allein durch den Aufbau der Sätze bestimmt ist.

Während die Dateneinheiten Feld, Segment und Satz aus ungleichen Untereinheiten aufgebaut sind bzw. sein können, besteht die Datei aus gleichartigen Untereinheiten, hier Datensätzen (DWORATSCHECK, 1971). Auf der Satzebene ist zwischen logischen und physischen oder physikalischen Sätzen (Record) zu unterscheiden. Nur bei konventionell organisierten Dateien, die eng umgrenzten Verarbeitungszwecken dienen, können Sätze in der Form gespeichert werden, wie sie auch wiedergewonnen und verarbeitet werden. In einer Datenbank dagegen müssen Sätze für viele Verarbeitungen, die zum Teil bei der Erstellung der Datenbank noch nicht bekannt sind, wiedergewonnen werden. Da die Verarbeitungsmöglichkeiten sehr zahlreich werden können, ist es nicht möglich, alle tatsächlich benötigten Sätze auch physisch zu speichern, was allerdings auch nicht notwendig ist, da benötigte Sätze aus jeweils unterschiedlichen Daten zusammengestellt werden können. Gespeichert werden die Daten daher nur in einem physischen Satz. Der logische Satz dagegen ist verarbeitungsorientiert und braucht mit dem physischen Satz nicht identisch zu sein. Der Benutzer der Datenbank arbeitet mit einem logischen Satz, der in Wirklichkeit aus mehreren physischen Sätzen oder Teilen davon in verschiedenen Dateien zusammengesetzt ist. Der logische Satz kann also Teile eines physischen Satzes enthalten, mit einem phy-

sischen Satz identisch sein oder mehrere physische Sätze oder Teile daraus enthalten, wobei diese physischen Sätze einer oder mehreren Dateien der Datenbank angehören können.

Die bisherige Identität der gespeicherten und verarbeiteten Sätze bedeutet im Grunde die Beschränkung auf eine einzige Datenstruktur.

Die Unterscheidung in logische und physische Sätze spiegelt sich wieder in den Begriffen Datenstruktur und Speicherstruktur (vgl. auch DWORATSCHECK, 1971). Wir definieren die obigen Begriffe für unsere Zwecke wie folgt:

Datenstruktur ist die externe Beschreibung der Daten für den Benutzer und dient der Darstellung der logischen Organisation der Daten.

Speicherstruktur ist die interne Darstellung der Daten, die der tatsächlichen Speicherung entspricht.

Die Aufgabe der Datenorganisation ist es nun, eine Speicherstruktur zu realisieren und eine Verbindung zur Datenstruktur herzustellen. Trennt man konsequent logischen und physischen Satz, so ist damit auch gegeben, daß die Zugriffsmöglichkeiten zu den Daten wesentlich erweitert werden. Allerdings müssen dann auch Verfahren eingeführt werden, die die Gewinnung von logischen Sätzen aus mehreren physischen Sätzen oder Teilen daraus ermöglichen. Ein Beispiel hierfür mag die Realisierung eines Stream-I/O für den Zugriff auf invertierte Listen in unserem Literaturprogrammsystem RS-II sein.

Zusammenhängend soll auf diese Methoden weiter unten bei "Such- und Änderungsverfahren" eingegangen werden.

Im folgenden soll zunächst beschrieben werden, wie logische Sätze aufgebaut sein können, ohne auf die physische Realisierung einzugehen.

Als einfachste Form ist hier zunächst der lineare Satz zu nennen, der sich durch die Gleichartigkeit aller seiner Felder auszeichnet. Felder gehören allein zum Ordnungsbegriff des Satzes und sind in sich nicht logisch unter- oder übergeordnet. Betrachtet man die Felder eines Satzes als Knoten und Beziehungen zwischen den Feldern als Kanten, so lassen sich lineare Sätze durch einen Graphen darstellen, dessen Knoten mit Ausnahme des ersten und des letzten Knotens jeweils einen Vorgänger und einen Nachfolger haben (KNUTH, 1968).

Lineare Sätze oder auch lineare Listen sind nach der Auffassung von WEDEKIND (1971) von der Datenorganisation her leicht zu handhaben und werden daher häufig realisiert. Sie enthalten aber keine Möglichkeiten, logische Über- bzw. Unterordnungen zwischen Feldern darzustellen (auf Satzebene). Enthalten die Sätze Segmente, also Zusammenfassungen logisch zusammengehöriger Felder, lassen sich auch diese durch lineare Graphen darstellen, wenn alle Segmente gleichgeordnet sind. Die Knoten sind dann nicht mehr Felder, sondern Segmente.

Bei nichtlinearen Sätzen können die Knoten mehr als einen Vorgänger oder Nachfolger haben. Hier ist eine Vielzahl von Strukturen denkbar, und entsprechend hoch ist die Zahl verschiedener Typen und Begriffe in der Literatur.

Als zwei Haupttypen von nichtlinearen Satzstrukturen sollen die hierarchische Struktur, insbesondere Baumstrukturen, und sonstige nichtlineare Strukturen, wie

z.B. zyklische Strukturen oder Netzstrukturen, herausgehoben werden.

Die wichtigsten hierarchischen Strukturen sind Baumstrukturen. Hierbei gibt es mehrere Hierarchie-Ebenen (Stufen), und die Anzahl der Knoten nimmt von oben nach unten je Stufe zu, wobei die erste Stufe immer nur einen Knoten hat. Weiter gilt für Bäume, daß jeder Knoten nur immer einen übergeordneten Knoten hat (vgl. CODASYL-System). Anders ausgedrückt heißt dies: zwei Knoten eines Baumes können immer nur durch einen Kantenzug verbunden werden (WEDEKIND, 1971). Ein Sonderfall eines Baumes ist ein sogenannter Binärbaum, bei dem jeder Knoten höchstens zwei untergeordnete Knoten besitzt (SALTON, 1968).

Sonstige nichtlineare Strukturen lassen beliebige Verbindungen zwischen den Knoten eines Satzes zu. Jeder Knoten kann mit jedem anderen Knoten durch eine Kante verbunden werden. Diese Strukturen werden als Networks oder Netzstrukturen bezeichnet. Stellt man Datenorganisation mit Hilfe von Graphen dar, so sind zwei Punkte besonders zu beachten:

1. Der Aufbau logischer und physischer Sätze kann sehr unterschiedlich sein. Bei der Beschreibung der Datenbank ist also zu unterscheiden, auf welcher Ebene welche logischen oder physischen Satzstrukturen möglich sind.

Beschreibt man logische Sätze, so ist anzugeben, aus welchen Elementen diese genau bestehen, da die Elemente ja physische Felder, Segmente oder Sätze sein können. Damit gewährleistet man Rückschlüsse auf die tatsächliche physikalische Speicherung.

2. Es ist genau zu unterscheiden zwischen der Darstellung eines logischen Aufbaus eines Satzes und der Darstellung des Zugriffs zur Gewinnung eines Satzes. WEDEKIND (1971) spricht in diesem Zusammenhang von logischen Graphen oder Zugriffsgraphen. Der Zugriffsgraph ist eine Darstellung der Schritte, die zur Gewinnung eines logischen Satzes notwendig sind, insbesondere der Suchschritte.

Es sei an dieser Stelle bemerkt, daß der wesentliche Nachteil des CODASYL-Konzepts darin besteht, daß bereits mit der Beschreibung der Daten(struktur) (DDL) auch die Zugriffsstruktur festgelegt ist.

Während ein logischer Graph ungerichtet ist, ist ein Zugriffsgraph immer gerichtet. Bei der Implementierung einer Datenbank sind grundsätzlich Analysen darüber anzustellen, für wieviele Anwendungen hierarchische Sätze nützlich sein können, denn es ist durchaus nicht immer sinnvoll, hierarchische Sätze physisch-hierarchisch zu speichern. Physisch-hierarchische Sätze erleichtern zwar die Beantwortung von Fragen, die dieser Hierarchie entsprechen, sie erschweren aber Beantwortungen von Fragen, die eine andere Hierarchie erfordern. Ein Aufbau von komplexeren hierarchischen physikalischen Sätzen ist nur dann sinnvoll, wenn diese zugrundeliegende Hierarchie auch der dominanten Anwendung entspricht, da dann der physische Satz so ausgelegt werden kann, daß er mit dem die Hauptverarbeitung bestimmenden logischen Satz übereinstimmt.

An ein Datenbanksystem ist daher die Forderung zu stellen, daß nicht nur nichtlineare Satzorganisation auf physikalischem Niveau zu unterstützen ist, sondern auch die Gewinnung beliebiger, unterschiedlich strukturierter,

nichtlinearer logischer Sätze. Aus dieser Forderung wird noch einmal die Bedeutung der Unterscheidung zwischen logischen und physischen Sätzen deutlich.

### 2.3 Darstellung von Daten durch Schlüssel

Es sei zunächst die Funktion eines Schlüssels erörtert. Bei formatierten Daten ist es weithin gebräuchlich, nicht-numerische Daten durch einen Code, einen Schlüssel oder eine Nummer zu speichern und zu verarbeiten. Die durch die Verschlüsselung erreichte wesentliche Verkürzung der Daten wird allerdings in Zukunft mehr und mehr an Bedeutung verlieren, da die Kostenfrage bei Speichern zunehmend in den Hintergrund tritt.

Eine weitere Funktion von Schlüsseln ist die Erleichterung bzw. die Ermöglichung von Kommunikation innerhalb von Organisationen. Zentraler Aspekt hierbei ist die Beseitigung der Redundanz der Umgangssprache, die um so notwendiger ist, je größer eine Gruppe bzw. ein Kommunikationsnetz wird, in dem Informationen ausgetauscht werden müssen.

Im folgenden werden die Hauptfunktionen eines Schlüssels aufgeführt.

1. Die wichtigste Funktion ist die der Identifikation. Es muß jederzeit möglich sein, einen Rückschluß auf die Originaldaten zu ziehen, d.h. die Originalinformation muß identifiziert werden können. Auf dem Satzniveau ist diese Funktion von besonderer Bedeutung, da Schlüssel, die als Ordnungsbegriffe verwendet werden, angeben, auf welches Objekt sich die im Satz gespeicherten Daten beziehen. Die Schlüssel des Ordnungsbegriffs identifizieren ein Objekt. Der Zusammen-

menhang zwischen beiden muß eindeutig sein. Da sich jeder Satz auf ein anderes Objekt bezieht, müssen sich die Schlüssel der Ordnungsbegriffe in jedem Satz unterscheiden. Erfüllt also ein Schlüssel nur eine der drei Funktionen Identifikation, Klassifikation und Information, so wird er nach der Hauptfunktion benannt.

2. Die Klassifikationsfunktion dient der Beschreibung eines zu verschlüsselnden Objektes, indem sie eine Einordnung in eine bestimmte Merkmalsklasse vornimmt. Man kann z.B. eine Personennummer aus bestimmten Zifferngruppen aufbauen, die ihrerseits jeweils eine bestimmte Bedeutung haben, so z.B. Personenkennziffer oder Identifikationsziffer. In diesem Fall enthält der Schlüssel eine Reihe von Informationen über das Objekt, d.h. über die betreffende Person oder den betreffenden Patienten. Klassifizierende Schlüssel reichen aber nicht aus, um das verschlüsselte Objekt eindeutig zu identifizieren, da einige Objekte in allen Ausprägungen der Merkmale in dem klassifizierenden Schlüssel übereinstimmen können. Da jeder Schlüssel eine Identifikationsfunktion erfüllen muß, ist es meistens notwendig, klassifizierende Schlüssel durch eine laufende Nummer zu ergänzen. Für den Ordnungsbegriff bzw. Hauptordnungsbegriff einer Datei werden häufig klassifizierende Schlüssel verwendet. In diesem Fall wird das verschlüsselte Objekt durch den Ordnungsbegriff nicht nur identifiziert, sondern der Ordnungsbegriff enthält auch weitere Daten über das Objekt. Die Nachteile des klassifizierenden Schlüssels liegen in seinem großen Stellenbedarf und in der Anfälligkeit bei Änderungen. Bei der Einführung zusätzlicher Klassifikationsmerkmale kann der gesamte Schlüssel gesprengt werden und muß geändert werden. Prinzipiell ist allerdings jede Änderung von Schlüsseln zu vermeiden, da sie eine erhebliche Umstellung bedeutet. Zusätzlich

wird die Identifikation der Objekte erschwert, da während des Umstellungsverfahrens parallele Verschlüsselungen vorkommen. Weiterhin kann es zu Änderungen kommen, wenn sich die Objekte selbst ändern und die Schlüssel stehen bleiben (z.B. Wohnungswechsel eines Patienten). Bei klassifizierenden Schlüsseln unterscheidet man, wie in Klassifikationssystemen auch, zwischen Begriffsleitern und Begriffsreihen oder auch hierarchischen und nebengeordneten Gliederungen.

3. Die Funktion der Information übernehmen die Informationsschlüssel oder auch sprechende Schlüssel, die ohne Benutzung einer Übersetzungsanleitung in der Lage sind, Informationen über ein bestimmtes Objekt abzugeben (z.B. laufende Nummer der Monatszahl, Autokennzeichen für Städte, Abkürzungen). Man darf klassifizierende und Informationsschlüssel nicht miteinander verwechseln. Es ist zwar möglich, daß innerhalb eines Klassifikationssystems die Signaturen für Einzelelemente sprechend, d.h. informativ sind, dies entbindet aber nicht von der Notwendigkeit der prinzipiellen Unterscheidung zwischen beiden Schlüsselarten. Ein Informationsschlüssel genügt dann, wenn nicht sehr viele Objekte vorhanden sind, ansonsten muß er ebenfalls durch einen Identifikationsschlüssel (z.B. laufende Nummer) ergänzt werden.

Ein Beispiel für eine Kombination der drei Funktionen Information, Klassifikation und Identifikation ist das Autokennzeichen.

Die Nachteile von Schlüsseln, die nur eine Funktion erfüllen, gleicht die Parallelverschlüsselung aus, bei der getrennte identifizierende und klassifizierende Schlüssel kombiniert werden. Das zu verschlüsselnde Objekt wird danach durch einen reinen Identifikations-

schlüssel identifiziert, der für das Objekt immer gleich bleibt. Alle klassifizierenden Beschreibungen dieses Objektes jedoch werden in einem eigenen Schlüssel erfaßt. Da der letztere beliebig ergänzt und verändert werden kann, ohne daß der Identifikationsschlüssel verändert wird, sind die Vorteile beider Schlüssel miteinander vereint.

Um die Änderungsfähigkeit von klassifizierenden Schlüsseln weiter zu verbessern, kann man einen Klassifikationsschlüssel so aufteilen, daß er nicht mehr als Ganzes ein Objekt beschreibt, sondern Teile des Schlüssels, die selbständig geworden sind, beschreiben Merkmale des Objektes.

Bei Personaldaten können z.B. die einzelnen Merkmale, die jeweils eigene Felder in Anspruch nehmen, sprechend (informierend) oder nicht-sprechend verschlüsselt werden, entsprechend den gegebenen Möglichkeiten. Für den Benutzer ist dieser Parallelschlüssel in Teilen angebar; er kann sich die Daten über das Objekt herausgeben lassen, die ihn interessieren, unter Verzicht auf für ihn irrelevante Information aus einem umfangreichen klassifizierenden und identifizierenden Schlüssel.

Die Parallelverschlüsselung vermeidet die Bildung von Hierarchien, da alle Schlüssel gleichberechtigt nebeneinander stehen, womit beliebige Schlüssel des Satzes für Auswertungen benutzt werden bzw. zu logischen Hierarchien gekoppelt werden können. Diese Möglichkeiten sind im Rahmen von Datenbanken von entscheidender Bedeutung.

Schließlich bietet die Parallelverschlüsselung günstige Voraussetzungen für die Zusammenfassung von getrennt bearbeiteten Datengebieten im Rahmen einer Vereinheitlichung von Schlüsseln (Medical record linkage, ACHESON, 1969).

Im Zusammenhang mit der Implementierung von Modellvorstellungen des Relationalmodells (Bildung von Normalformen) hat der Schlüsselbegriff an Bedeutung gewonnen, wie weiter unten gezeigt werden soll.

### 3 Methoden der Daten(bank)organisation

Bei den Methoden der Datenorganisation sollte grundsätzlich zwischen Speicherungsverfahren einerseits und Such- und Änderungsverfahren andererseits unterschieden werden. Bei der konventionellen Datenorganisation mit einer starken Abhängigkeit zwischen Speicherung und Suche wird diese Unterscheidung selten getroffen. Meist findet man diese Verfahren unter einem gemeinsamen Oberbegriff (Speicherung und Zugriffsform (DWORATSCHECK, 1971), oder Speicherungs- und Verarbeitungsformen (AHRENS, WALTER, LUTZ, KLIMESCH, 1971)).

Eine Unterscheidung zwischen beiden Verfahrensgruppen ist deshalb wichtig, weil, wie in der Einführung bereits gezeigt, die Speicherung und die Verarbeitung der Daten streng getrennt werden sollten und die Verbindung zwischen beiden über die Suchverfahren hergestellt werden sollte. Dies bedeutet z.B. bei einer sequentiellen Speicherung von Daten in einer Datenbank weder, daß diese Daten auch immer sequentiell aufgesucht, noch daß sie sequentiell verarbeitet werden müssen, wie es bei konventioneller Anwendung der EDV überwiegend der Fall ist. Daher konnte man dort die Begriffe der sequentiellen Speicherung und der sequentiellen Suche fest zuordnen oder durch einen Begriff ersetzen.

§

#### 3.1 Generelle Speicherungsmöglichkeiten

Für die folgende Darstellung der genannten Speicherungsmöglichkeiten wird ein Speichermedium mit linearem

(unendlichem) Adressraum vorausgesetzt. Über die physikalische Realisierung werden keine Angaben gemacht. Obwohl oben ausdrücklich eine Unterscheidung zwischen logischen und physikalischen Sätzen gefordert wurde, soll im folgenden darauf verzichtet werden und beides aus Gründen der anschaulicheren Darstellbarkeit gleichgesetzt werden.

Es wird folgende Symbolik verwendet:

 = (relative) Speicheradresse (in Ausnahmefällen auch Inhalt eines Satzes)

 = (logischer) Satz mit Werten  
(VALUE, Daten)

 = Schlüssel (Ordnungsbegriff(e))

Später wird eingeführt:

 LINK = Zeiger (Pointer, Link)

Beispiel:





### 3.1.1 Fortlaufende Speicherung

Bei der fortlaufenden Speicherung werden die Speicherplätze eines Speichermediums lückenlos, beginnend beim ersten Speicherplatz, belegt. Nicht benötigte Speicher-

plätze bleiben frei, so daß Plätze mit den höchsten Speicheradressen immer frei sind. Die Anordnung der Sätze erfolgt nach einem bestimmten Ordnungsprinzip. In Abhängigkeit davon lassen sich die Verfahren der seriellen Speicherung und der sequentiellen Speicherung, sowie die Speicherung nach der Abfragehäufigkeit unterscheiden.

Bei der starr fortlaufenden Speicherung (serielle Speicherung, AHRENS & WALTER, 1971) werden die Sätze so, wie sie anfallen, gespeichert, d.h. allein nach dem Ordnungsprinzip der zeitlichen Folge. Eine sich daraus ergebende Sortierung der Sätze ist für die Wiedergewinnung der Daten nur selten von Nutzen. Wird ein bestimmter Satz benötigt, so muß die Datei Satz für Satz von Anfang an durchgesucht werden. Aus diesem Grund wird die serielle Speicherung bisher in der Praxis nur selten verwendet.

Mit der Einführung neuer Suchverfahren, die einen besseren Zugriff ermöglichen, auch wenn die Datei seriell gespeichert ist, hat diese Speicherungsform jedoch wieder an Bedeutung gewonnen, da die Speicherung der Daten und die Datenverwaltung (neu hinzukommende Sätze werden am Ende der Datei angefügt) sehr einfach sind.

Beispiel für eine serielle Speicherung:

|    |      |
|----|------|
| 17 | 3.09 |
|----|------|

①

|   |      |
|---|------|
| 4 | 2.37 |
|---|------|

②

|    |      |
|----|------|
| 33 | 3.11 |
|----|------|

③

|    |      |
|----|------|
| 11 | 4.68 |
|----|------|

④

|    |      |
|----|------|
| 19 | 1.57 |
|----|------|

⑤

Bei der zweiten Form der fortlaufenden Speicherung, der sequentiellen Speicherung, die auch logisch fortlaufende Speicherung genannt wird (AHRENS & WALTER, 1971), bestimmt der Ordnungsbegriff die Anordnung der Sätze. Jeder gespeicherte Satz hat bei aufsteigender Sortierung einen höheren Ordnungsbegriffswert als sein Vorgänger. Diese logische Folge kann das Aufsuchen der Sätze erleichtern, wenn die Ordnungsbegriffe der gesuchten Sätze bekannt sind.

Das bisher am häufigsten angewendete Speichungsverfahren ist die sequentielle Speicherung. Ihr Hauptnachteil ist die fehlende Möglichkeit, einen Satz zu entfernen oder einzufügen, ohne daß alle folgenden Sätze neu geschrieben werden müssen (HARTMANN, MATTHES & PRÖME, 1969).

Die Verschiebung der Folgesätze ist jedoch notwendig, wenn entweder ein Platz frei geworden ist oder ein neuer Platz benötigt wird. Man kann allerdings beim Löschen der Sätze auf das physikalische (tatsächliche) Entfernen dieser Sätze verzichten und diese nur als gelöscht, d.h. logisch nicht vorhanden, kennzeichnen. Dazu wird allerdings wieder zusätzlicher Speicherplatz benötigt.

Beispiel für eine sequentielle Speicherung:

|   |      |
|---|------|
| 4 | 2.37 |
|---|------|

1

|    |      |
|----|------|
| 11 | 4.68 |
|----|------|

2

|    |      |
|----|------|
| 17 | 3.09 |
|----|------|

3

|    |      |
|----|------|
| 19 | 1.57 |
|----|------|

4

|    |      |
|----|------|
| 33 | 3.11 |
|----|------|

5

Die fortlaufende Speicherung kann durch Verwendung der Abfragehäufigkeit als Ordnungsprinzip für die Anordnung der Sätze durchgeführt werden. Der am häufigsten benötigte Satz wird als erster Satz, der am zweithäufigsten benötigte Satz als zweiter Satz gespeichert, und so fort (SALTON, 1968; LIMBURG & MACHMER, 1970). Für häufig benötigte Sätze ergeben sich dabei geringe Suchzeiten, die allerdings für selten benötigte Sätze sehr in die Höhe schnellen. Hauptnachteil des Speicherverfahrens ist die vorher notwendige Auszählung der Abfragehäufigkeiten, Diese sind in den meisten Fällen zum Zeitpunkt des Einspeicherns nicht bekannt. Es ist deshalb notwendig, zunächst eine andere Datei aufzubauen, um aus dieser eine nach der Abfragehäufigkeit organisierte Datei neu anlegen zu können. Bei der Verwendung dieser Datei über einen Zeitraum hinweg kann man Abfragehäufigkeiten je Satz festhalten und die Datei nach bestimmten Zeitabschnitten nach diesen ermittelten Häufigkeiten reorganisieren. Der Aufwand für dieses Verfahren ist allerdings nur zu empfehlen für Dateien, in denen die Satzzahl konstant bleibt, d.h. bei denen keine neuen Sätze hinzukommen, denn für die neu hinzukommenden Sätze müßten erneut Abfragehäufigkeiten ermittelt werden.

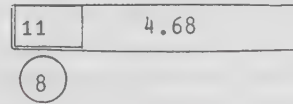
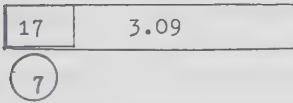
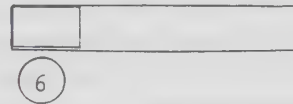
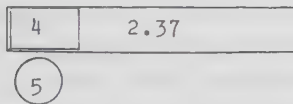
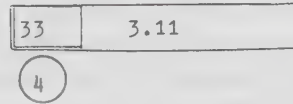
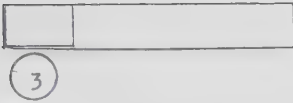
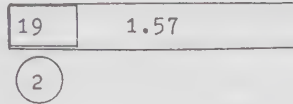
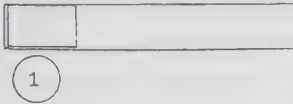
Eine weitere Schwierigkeit ergibt sich bei der Benutzung der Datenbank durch mehrere Datenbankbenutzer, für die jeweils unterschiedliche Abfragehäufigkeiten je Satz gelten dürften, die zudem über einen bestimmten Zeitraum hinweg nicht konstant bleiben.

### 3.1.2 Gestreute Speicherung

Sätze bei der gestreuten Speicherung sind ohne Verwendung eines Ordnungsprinzips auf dem Speichermedium ange-

ordnet, wodurch Lücken in der Belegung der Speicherplätze auftreten können.

Beispiel für eine gestreute Speicherung:



Eine Änderung der Datei mit gestreutem Speicherungsmodus läßt sich gut durchführen, da ein einmal gespeicherter Satz nicht mehr verschoben werden muß, wenn er in die Datei neu aufgenommen wird.

Wir unterscheiden bei der gestreuten Speicherung zwei Anwendungsfälle.

- a) Es ist ausschlaggebend, daß ein bestimmter Speicherplatz frei ist. Dieser Platz auf dem Speicher wird durch eine Lückentabelle gekennzeichnet, in der die noch freien Stellen verzeichnet sind (MÜLLER, 1970; KNUTH, 1973). Handelt es sich um Sätze variabler Länge, muß dabei beachtet werden, daß ein Speicherplatz ausgewählt wird, bei dem der benötigte und freie Speicherplatz am besten übereinstimmen, der freie Speicherplatz also nicht oder nur wenig größer ist als der benötigte. Man nennt dieses Verfahren die best-fit-Methode im Gegensatz zur first-fit-Methode, bei der der erste ausreichend große freie Speicherplatz ausgewählt wird (KNUTH, 1973). Diesem Speicherungsverfahren der gestreuten Speicherung mit Lückentabelle sei Absatz b) gegenübergestellt.
- b) Hier wird aus dem Ordnungsbegriffswert eines zu speichernden Satzes mit Hilfe eines Algorithmus ein Speicherplatz errechnet. Dieses Vorgehen, das indirekte Adressierung genannt wird, dient neben der Speicherung der Sätze auch der Suche und wird deshalb weiter unten bei den Suchverfahren behandelt.

Oft wird in der Literatur gestreute Speicherung mit der häufiger verwendeten indirekten Adressierung gleichgesetzt. Auch diese Gleichsetzung ist letztlich auf die Tatsache zurückzuführen, daß Speicherungsverfahren und Suchverfahren logisch nicht streng getrennt werden. Bei der "gestreuten Speicherung mit indirekter Adressierung" kann der Fall eintreten, daß ein Platz, der für einen zu speichernden Satz bereitgestellt werden soll, bereits durch einen anderen Satz, dessen Ordnungsbegriffswert zu demselben Speicherplatz führt, belegt ist. In diesem Fall wird dieser Satz in einem anderen Speicherbereich abgelegt, wobei allerdings an dem errechneten Speicher-

platz in einem besonderen Feld vermerkt wird, wo der dort eigentlich hingehörende Satz dann tatsächlich steht. Man spricht dann von Überlaufsätzen, sogenannten Overflow-records. Der Speicherbereich, in dem die Überlaufsätze gespeichert sind, wird als Überlaufbereich bezeichnet; er enthält Sätze, die im sogenannten Primärbereich keinen Platz gefunden haben (HARTMANN, MATTHES & PROEME, 1968).

### 3.1.3 Kombination von fortlaufender und gestreuter Speicherung

Mit Hilfe einer Kombination beider Speicherungsprinzipien kann man den Hauptnachteil der sequentiellen Speicherung vermeiden, nämlich beim Einfügen neuer Sätze die Folgesätze verschieben zu müssen. Die Datei wird zunächst bei der Umladung sequentiell gespeichert. Neu hinzukommende Sätze werden dann nicht in diese Datei eingefügt, sondern sie werden auf einen freien Speicherplatz oder in einen speziellen Überlaufbereich gebracht.

Kommt z.B. ein Satz nach seinem Ordnungsbegriff zwischen zwei Sätzen zu stehen, kann aber nicht, da hier kein Platz vorhanden ist, abgesetzt werden, so wird er im Überlaufbereich gespeichert, und im vorangehenden oder folgenden Satz wird vermerkt, wo der logische Nachfolger bzw. Vorgänger gespeichert ist.

Dieses Verfahren ist vor allem dann angebracht, wenn die Zahl neu hinzukommender Sätze nicht zu groß wird, d.h. nicht zu viele Überlaufsätze entstehen.

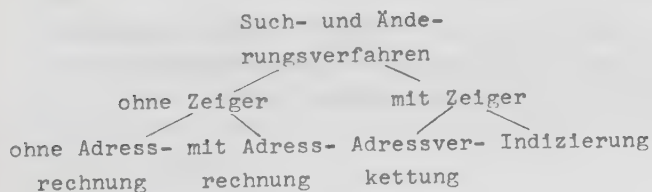
Auch bei einer zunächst seriell aufgebauten Datei läßt sich die gestreute Speicherung anwenden. Hier wird das Hauptaugenmerk darauf gelegt, daß bei Löschung von Sätzen die Auffüllung des frei gewordenen Platzes nicht durch Verschieben der Folgesätze geschieht, sondern daß stattdessen dieser freie Platz in einer Lückentabelle vermerkt wird. Neue Sätze werden nicht am Ende der Datei, sondern in die vorhandenen Lücken eingespeichert.

Kombinationen von fortlaufender und gestreuter Speicherung werden immer mehr an Bedeutung gewinnen, da sie erstens ein einfaches Prinzip beim Aufbau der Datei anwenden und zweitens dennoch ein leicht zu handhabendes und flexibles Vorgehen bei Änderungen der Datei ermöglichen.

### 3.2 Such- und Änderungsverfahren mit linearen Listen

Die Verbindung zwischen Speicherungs- und Verarbeitungsverfahren nennt man Suchverfahren. Sie dienen dazu, eine optimale Datenverarbeitung zu gewährleisten, obwohl mit der Einführung der Datenbank Datenspeicherung und Datenverarbeitung getrennt worden sind (CHAPIN, 1969). Mit zunehmender Umfangserweiterung und Kompliziertheit von Datenbanken ist auch die Abfrage, d.h. das Auffinden von bestimmten Informationen unter zeitlichem Aspekt problematischer geworden. Die Leistungssteigerung kann nicht nur durch physikalische Entwicklungen erbracht werden, vielmehr ist auch eine Entwicklung neuer Suchverfahren zur Beschleunigung von Suchprozessen notwendig. Suchverfahren sollte man nicht gleichsetzen mit Zugriffsverfahren, da z.B. ein Suchvorgang mehrere Zugriffe erforderlich machen kann, bis ein betreffender Satz gefunden worden ist. Der Maßstab für die Qualität eines

Suchverfahrens ist die Anzahl der benötigten Zugriffe. Es gibt bisher wenig vollständige Darstellungen von Suchverfahren (WEDEKIND, 1970). Wir folgen einem Vorschlag von MERTEN (1974) zur Darstellung von Suchverfahren, indem wir die Begriffe "Suchverfahren ohne Zeiger" und "Suchverfahren mit Zeigern" oder Pointern verwenden. Bei dem erstgenannten Verfahren werden Suchbegriffe oder Suchargumente unmittelbar mit gespeicherten Daten verglichen, während bei dem zweiten Verfahren neben den eigentlichen Daten weitere Daten gespeichert werden, die überhaupt erst eine Suche ermöglichen oder sie erleichtern. Man nennt diese Daten, die bestimmte Feldwerte und Adressen von Sätzen enthalten können, "Indiceszeiger" oder "Pointer". Mit diesen Zeigern werden die Suchargumente verglichen. Bei Gleichheit verweisen die Zeiger auf angegebene Adressen und gesuchte Sätze.



In der vorhergehenden Abbildung wird dargestellt, wie man beide Verfahren jeweils wieder unterteilen kann, und zwar die Suchverfahren ohne Zeiger in solche ohne Adressrechnung und solche mit Adressrechnung, die Suchverfahren mit Zeiger in solche, die mit Adressverkettung arbeiten, und solche, die mit Indizierung arbeiten (LUTZ & KLIMESCH, 1971).

Bei Suchverfahren ohne Adressrechnung muß mehrfach auf gespeicherte Sätze zugegriffen werden, bis der eigentliche Satz gefunden worden ist. Bei Suchverfahren mit

Adressrechnung gibt es Möglichkeiten, aus dem Suchargument einen Speicherplatz zu errechnen, auf dem der gewünschte Satz steht bzw. stehen sollte. Erst dann wird unmittelbar auf diesen Satz zugegriffen.

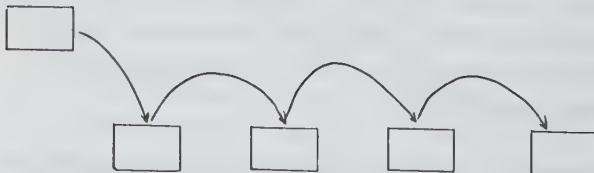
Bei der Adressverkettung werden die Zeiger mit den zu speichernden Daten, also den Primärdaten, gespeichert. Bei dem Verfahren mit Indizierung werden Zeiger selbst in eigenen Dateien, die also nur Sekundärdaten zu Suchzwecken enthalten, gespeichert. Im folgenden werden die Suchverfahren im Einzelnen beschrieben.

### 3.2.1 Suchverfahren ohne Zeiger

#### a) Suchverfahren ohne Adressrechnung

##### 1) Sequentielle Suche

Bei der sequentiellen Suche (lineare Suche, sukzessive Suche) wird eine Datei Satz für Satz durchsucht, bis der betreffende Satz gefunden worden ist.



Zur Bewertung eines Suchverfahrens ist allgemein die Angabe der Zeitdauer heranzuziehen, die im Durchschnitt zum Auffinden eines Satzes gebraucht wird. Da diese jedoch von verschiedenen äußeren Faktoren abhängt, empfiehlt sich als Bewertungskriterium die mittlere Zugriffszahl (WEDEKIND, 1970). Es ist dies die Kennzahl, die ausgedrückt wird durch das arithmetische Mittel aller Anzahlen von Zugriffen, wenn ein Satz aus einer bestimmten Datei herausgefunden werden soll. Sie sagt aus, wie oft im Durchschnitt auf diese Datei zugegriffen werden muß, bis ein durch ein Suchargument definierter Satz gefunden ist. Wir wollen im folgenden diese Zugriffszahl als Bewertungskriterium verwenden. Das physikalische Speichermedium selbst liefert nur einen Faktor, mit dem die mittlere Zugriffszahl zu multiplizieren ist, um die Zugriffszeit zu erhalten.

Ist eine Datei mit  $n$  Sätzen gegeben, die  $z_i$  Zugriffe benötigt, um einen bestimmten Satz  $i$  aufzufinden, und  $p_i$  sei die Wahrscheinlichkeit, mit der der Satz  $i$  gesucht wird, dann ist die mittlere Zahl von Zugriffen, also  $\bar{z}$ , der Erwartungswert für die Anzahl der Zugriffe bei einer Suche.

$$\bar{z} = \sum_{i=1}^N z_i p_i$$

Im konkreten Fall werden die mittleren Zugriffshäufigkeiten (die Zahl der in einem bestimmten Zeitraum zu einem Satz erfolgten Zugriffe, bezogen auf die Gesamtzahl der Zugriffe auf die Datei in diesem Zeitraum) kaum ermittelt werden können.

Es wird normalerweise vielmehr ein Modell der Gleichverteilung angenommen; in diesem Fall wandelt sich die obige Formel wie folgt um

$$p_i = \frac{1}{N} \quad \bar{z} = \frac{1}{N} \sum_{i=1}^N z_i$$

Da bei der sequentiellen Suche gilt  $z_i = 1$ , gilt ferner

$$\bar{z} = \frac{1}{N} \sum_{i=1}^N 1 = \frac{1}{N} \frac{(N+1)N}{2} = \frac{N+1}{2}$$

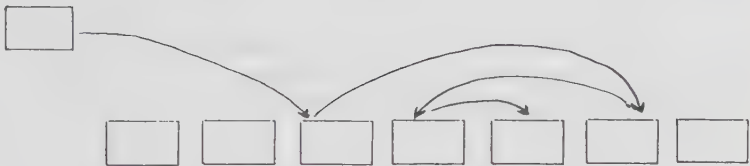
Dies bedeutet, daß im Schnitt etwa die halbe Datei durchsucht werden muß, um einen betreffenden Satz zu finden. Das sequentielle Suchverfahren hat also die höchste mittlere Zugriffszahl. Sie ist nur in dem Fall günstig, wenn mehrere Sätze gleichzeitig benötigt werden, d.h. wenn mehrere Argumente vorliegen und die Suchargumente sortiert werden können (FLORES, 1970). Ist ein betreffender Satz gefunden, so kann er nur Suchargumente und Ordnungsbegriffe in einer gemeinsamen Folge haben. Der nächste zu findende Satz kann nur noch im nichtdurchsuchten Teil der Datei vorhanden sein. Es ist daher nicht erforderlich, nach jedem gefundenen Satz wieder von vorn, d.h. am Anfang der Datei zu beginnen.

## 2) m-Wege-Suche

Beim m-Wege-Suchen, auch Sprungverfahren oder serielles Suchen mit gleichmäßigem Sprung genannt, wird die zu durchsuchende Datei in Blöcke unterteilt, wobei in allen Blöcken dieselbe Anzahl von Sätzen enthalten sein

muß. Bei der Suche nach einem bestimmten Satz wird zunächst festgestellt, in welchem Block sich dieser Satz befindet. Ist das Suchargument größer als das des betreffenden Satzes, so wird der nächste Block geprüft. Ist es kleiner, so ist der richtige Block gefunden. Im richtigen Block wird dann sequentiell der gewünschte Satz aufgesucht. Ist das Suchargument gleich, so ist bereits der richtige Satz gefunden.

Dieses Verfahren sei an einer kurzen Abbildung erläutert. Es handelt sich hier um drei Blöcke à 5 Sätze; gesucht ist der Satz mit dem Ordnungswert 13.



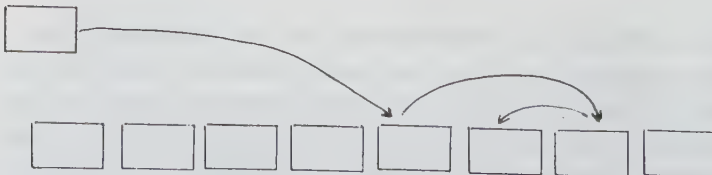
Voraussetzung der m-Wege-Suche ist das Vorliegen einer sequentiellen Datei, in der nach einem Wert des Ordnungsbegriffes gesucht wird. Es ergibt sich die mittlere Zugriffszahl als  $\sqrt[n]{n}$ . Bei optimaler Blocklänge sind die Anzahl der Blöcke sowie die Anzahl der Sätze je Block und die mittlere Zugriffszahl gleich (SIX, 1973; JANKO, 1976).

Der Wert  $n$  für die mittlere Zugriffszahl ist wesentlich günstiger als der entsprechende Wert bei der sequentiellen Suche, was sich bei großem  $n$  verstärkt auswirkt (SALTON, 1968).

### 3) Binäre (duale) Suche

Die binäre Suche ist wie die  $m$ -Wege-Suche in ihren Anwendungsmöglichkeiten eingeschränkt, da sie ebenfalls nur bei Suchvorgängen in einer sequentiellen Datei unter Angabe eines Ordnungsbegriffswertes durchführbar ist. Bei dieser Art der Suche wird in der Mitte der zu durchsuchenden Datei begonnen; es wird also der Wert des Ordnungsbegriffes des Satzes mit der Nummer  $n/2$  oder  $n+1/2$  mit dem Suchargument verglichen. Ist der Wert gleich, so ist der Satz gefunden. Ist der Wert des Suchargumentes kleiner (bzw. größer), so muß der gesuchte Satz in der ersten (bzw. zweiten) Hälfte der Datei liegen. Die verbleibende Hälfte wird wiederum halbiert. Dieser Prozeß wird solange fortgeführt, bis der gewünschte Satz gefunden ist.

Bei der binären Suche wird also bei jedem Zugriff entweder der gewünschte Satz gefunden oder die Zahl der noch zu durchsuchenden Sätze ungefähr um die Hälfte verringert.



Auch die binäre Suche sei an einem Beispiel erklärt. Der Zugriffssatz bezieht sich auf 15 Sätze. Gesucht ist der Satz mit dem Ordnungsbegriffswert 13. Bei der binären Suche können wir  $j$  ( $1 < j < a$ ) Zugriffsstufen unterscheiden. Um zu einer Stufe  $j$  zu gelangen, werden  $j$  Zugriffe benötigt. Auf jeder Zugriffsstufe können wiederum  $2^{j-1}$  Sätze erreicht werden. Die Summe der Anzahlen von Zugriffen, um alle Sätze einer Stufe  $j$  aufzusuchen, sei  $z_j$  bezeichnet, und es gilt  $z_j = 2^{j-1} \cdot j$ . Von einer idealen binären Suche (BERZTISS, 1975) spricht man, wenn folgende Beziehung gilt:  $a = \log_2 (n-1)$ . Ist  $N$  nicht eine um 1 verminderte Potenz von 2, so ist die Zahl der Sätze auf der Stufe  $a$  kleiner als  $2^{a-1}$ , und auch die obige Beziehung gilt nicht. In diesem Fall ist  $a$  der um 1 verminderte Logarithmus dualis der größten in  $N$  enthaltenen Potenz der Zahl 2, und es gilt:

$$N \approx \sum_{j=1}^{a-1} 2^{j-1}$$

Für den Fall der nichtlinearen Suche gilt für die mittlere Zugriffszahl

$$\bar{z} = 2^{\log N}$$

Dabei werden für alle Sätze gleiche Zugriffswahrscheinlichkeiten angenommen.

Somit ist die binäre Suche das schnellste der bisher beschriebenen Suchverfahren. Es wird deshalb auch in der Praxis häufig benutzt. Die abgeleitete mittlere Zu-

griffszahl gilt allerdings nur für das ideale binäre Suchen ( $n$  ist die um 1 verminderte Potenz von 2). Diese Bedingung ist für großes  $n$  nur sehr schwer einzuhalten. Dennoch kann in diesen Fällen die binäre Suche wirkungsvoll durchgeführt werden ohne eine wesentliche Beeinträchtigung der mittleren Zugriffszahl (LIMBURG & MACHNER, 1970).

b) Suchverfahren mit Adressrechnung

Bei diesem Verfahren kann aus den Suchargumenten die Adresse des Platzes ermittelt werden, an dem der gesuchte Satz steht (bzw. stehen müßte). Es liegt ein logischer Zusammenhang zwischen dem Suchargument und dem Speicherplatz des gesuchten Platzes vor. Der Speicherplatz ist eine Funktion des Sucharguments und damit zugleich des Feldinhalts des Satzes, da sich das Suchargument immer auf den Inhalt eines Feldes bezieht. Es ist praktisch (bucket size) keine Suche nötig, sondern nur eine Berechnung des Zusammenhangs zwischen Suchargument und Adresse. In den meisten Fällen wird bereits nach einem einzigen Zugriff der gewünschte Satz gelesen.

Man unterscheidet die Verfahren der direkten und die Verfahren der indirekten Adressierung. Beide Verfahren beziehen sich zunächst, wie das m-Wege-Suchen und die binäre Suche, nur auf Suchvorgänge unter Angabe eines Ordnungsbegriffswertes als Suchargument.

1) Direkte Adressierung

Bei dem Verfahren der direkten Adressierung gibt es

einen eindeutigen Zusammenhang zwischen dem Ordnungsbegriff eines Satzes und dem Speicherplatz des Satzes. Der Zusammenhang zwischen Ordnungsbegriff und Adresse ist umkehrbar, was im allgemeinen als das Kriterium für die direkte Adressierung angesehen wird. Wegen dieses Zusammenhangs sind die Sätze bei der direkten Adressierung immer sequentiell in der Folge des Ordnungsbegriffes gespeichert. Der Ordnungsbegriff selber kann wiederum in verschiedene Teile aufgeteilt werden, die sich z.B. beziehen können auf Magnetplattenstapel, Nummer des Zylinders, Nummer der Spur u.a.

Speicherplatz wird dann optimal ausgenutzt, wenn der Nummernkreis des Ordnungsbegriffes voll belegt ist, also für jede Nummer ein Satz vorhanden ist, und wenn die Länge der Sätze genau dem Speicherplatz zwischen zwei Adressen entspricht. Ist die Größe der Sätze höher oder niedriger als der verfügbare Speicherplatz pro Adresse, so kann mit Hilfe eines einfachen Algorithmus die errechnete Adresse erweitert oder verdichtet werden. Die umkehrbare Beziehung zwischen Ordnungsbegriff und Adresse bleibt dennoch erhalten.

Der Ordnungsbegriff wird zweckmäßigerweise als numerischer Begriff verwendet (identifizierende Schlüssel, fortlaufende Zähl-Nummer), wie er z.B. bei der Parallelverschlüsselung als Ordnungsbegriff benutzt wird. Solche Schlüssel haben bei direkter Adressierung zusätzlich die Funktion der Speicherplatzbestimmung.

Diese Suchverfahren, dessen mittlere Zugriffszahl gleich 1 ist und das eine Gesamtaktivitätsrate von  $n/N$  von ebenfalls 1 aufweist, hat gegenüber den bisher beschriebenen Suchverfahren die niedrigste Anzahl von Zugriffen. Es

begünstigt, im Gegensatz zu den meisten Suchverfahren, die Sofortverarbeitung und die Stapelverarbeitung. Allerdings müssen sich die Suchargumente auf den Ordnungsbegriff der Datei beziehen.

Wird bei der direkten Adressierung auch noch sequentiell gespeichert, so bedeutet dies, daß bei mechanischen Zugriffen z.B. der Plattenarm nicht hin und her bewegt werden muß, sondern sich fortlaufend bewegen kann (Fahrstuhlstrategie), wenn alle Sätze in der Sortierfolge des Ordnungsbegriffes benötigt werden. Die Zugriffszahlen können in beiden Fällen gleich sein. Die Zugriffszeiten unterscheiden sich aber wesentlich zugunsten der direkten Adressierung (WEDEKIND, 1970).

- 2) Indirekte Adressierung (Randomising oder Hashing)  
(MORRIS, 1968; MAURER & LEWIS, 1975)

Häufig sind die Werte eines Ordnungsbegriffes nicht für eine direkte Adressierung geeignet. Es können z.B. die Nummernkreise der Ordnungsbegriffe Sprünge aufweisen, wie dies z.B. bei der Verwendung eines klassifizierenden Schlüssels der Fall ist, oder der Nummernkreis hat zwar keine Sprünge, aber die tatsächlich vorhandenen Werte sind nur spärlich vorhanden. So können z.B. von 1.000 möglichen Sätzen nur 200 vorhanden sein und die Ordnungsbegriffswerte über den gesamten Nummernkreis streuen, oder die Ordnungsbegriffe sind keine numerischen Begriffe.

In all diesen Fällen muß man Algorithmen verwenden, die aus den Ordnungsbegriffen Speicherplätze errechnen. Diese Verfahren werden als indirekte Adressierung bezeichnet.

Die verwendeten Algorithmen sollen eine gleichmäßige Verteilung von Zahlen über einen bestimmten Wertebereich erzeugen, wobei jede Zahl einer Speicheradresse entspricht. Eingangswerte für diesen Algorithmus sind Ordnungsbegriffswerte der zu speichernden Sätze.

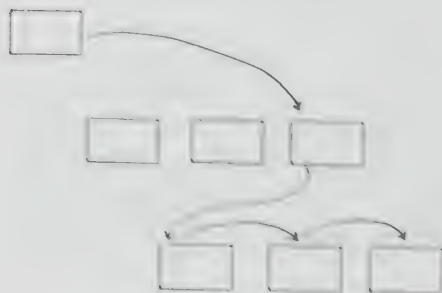
Ein Schlüssel habe z.B. die möglichen Werte  $x (1 < x < z)$ . Tatsächlich treten aber nur  $n$  Werte auf, die sich ausserdem nur auf bestimmte Bereiche konzentrieren. Es wird ein Algorithmus gesucht, der den Werten  $x$  Werte  $y (a < y < u)$  zuordnet, so daß gilt  $y = F(x)$ . Die Werte sollen als Adresse verwendet werden,  $a$  entspricht der Anfangsadresse des Speichers,  $u$  der letzten verfügbaren Adresse. Die Zahl der möglichen Werte von  $y$  soll im Idealfall gleich  $n$  sein. Es soll also möglichst eine lückenlose Belegung der Speicherplätze erfolgen. Auch bei ungleichmäßiger Verteilung der Werte von  $x$  soll eine gleichmäßige Verteilung der Werte von  $y$  entstehen, was in der folgenden Abbildung schematisch dargestellt wird.



Tatsächlich ist die ideale Belegung der Werte von  $y$  nicht zu errechnen; es bleiben immer einige Adressen frei; andere hingegen werden mehrfach berechnet.

In diesem zweiten Fall wird der zu speichernde Satz dann mit Hilfe einer Überlauftechnik gespeichert, und an dem errechneten Platz im sogenannten Primärbereich (dort, wo er eigentlich hingehört) wird ein Vermerk angebracht, der Auskunft darüber gibt, wo der betreffende Satz im sogenannten Überlaufbereich gespeichert wird. Kommt eine nochmalige Belegung der gleichen Stelle vor, so wird jetzt im ersten Überlaufsatz vermerkt, wo sich der zweite Überlaufsatz befindet.

Die indirekte Adressierung sei nochmals durch Darstellung eines sogenannten Zugriffsgraphen verdeutlicht. Es seien wiederum 15 Sätze mit verschiedenen Ordnungsbegriffen gespeichert; einige davon sind im Überlaufbereich gespeichert, weil die für sie errechneten Adressen bereits belegt waren.



Da bei indirekter Adressierung stets Überlaufsätze entstehen, werden für einige Sätze mehr als ein Zugriff in der Suche benötigt. Daher sollte die Zahl der Überlaufsätze möglichst niedrig gehalten werden. Diese Forderung steht aber in Widerspruch zu der Forderung nach möglichst vollständiger Belegung des zur Verfügung stehenden Speicherplatzes.

Werden  $a$  verschiedene Adressen geliefert, die für die Speicherung von  $n$  Sätzen bereitstehen, aber nicht alle von diesen  $a$  Adressen genau einmal errechnet, so bleiben Adressen frei. Je größer  $a$  im Verhältnis zu  $n$  ist, um so größer ist die Wahrscheinlichkeit, daß viele Sätze unbelegt bleiben und wenige Sätze doppelt belegt bleiben, was größeren Speicherplatzverbrauch zur Folge hat. Bezeichnet man die mehrfach belegten bzw. -errechneten Adressen mit  $q$ , so wird die Relation  $q/a$  als Belegungsfaktor (loading factor) bezeichnet. Je größer der Belegungsfaktor ist, desto mehr Überlaufsätze wird es geben, und umgekehrt (LUM, YUEN & DODD, 1971).

Man kann die Zahl der Überlaufsätze verringern, indem man die als Ergebnis des Umrechnungsalgorithmus möglichen Adressen verringert und für jede Adresse mehrere Sätze bereithält. Diese Speicherplätze bezeichnet man dann als Blöcke und die Zahl der Sätze pro Block als Blockungsfaktor. Erst wenn ein solcher Block voll belegt ist, kommt es zu Überlaufsätzen. Bei Anwendung der Blockung wird die Zahl der möglichen aus dem Umrechnungsalgorithmus entstehenden Adressen in etwa umgekehrt proportional zum Blockungsfaktor verringert. Bestehen die Werte eines Ordnungsbegriffs nicht aus numerischen Zeichen, so kann man durch Transformations-

vorschriften eine numerische Darstellung erreichen (z.B. Transformation des Alphabets in die Zahlen von 1 bis 26).

Im folgenden seien die wichtigsten Adressrechnungs-algorithmen kurz beschrieben.

#### - Stellenauswahl

Hier werden einzelne Stellen des Ordnungsbegriffs ausgewählt, die dann zusammen eine Adresse bilden. Man sollte möglichst Stellen auswählen, die eine große Streuung haben, z.B. solche, bei denen alle Ziffern 0 bis 9 vorkommen, so z.B. bei Verwendung von Datums-teilen die Einerstellen. Außerdem sollte darauf geachtet werden, daß die einzelnen Ziffern möglichst gleich-verteilt sind. Bei diesem Verfahren sind Vorauszählungen nötig.

#### - Das Falten

Hier wird der Ordnungsbegriff in Teile zerlegt, d.h. in Teilzahlen, deren Addition die Adresse ergibt. Dieses Verfahren ist sehr einfach anzuwenden, man erreicht aber keine gleichmäßige Belegung des Speicherplatzes, wenn mehrere Stellen des Ordnungsbegriffs eine geringe Streuung haben.

#### - Multiplikation

Ähnlich geht man vor, wenn man Teile des Ordnungsbegriffs miteinander multipliziert. Aus den entstehenden Produkten wählt man meist die mittleren Stellen als Adresse aus, da diese am meisten streuen.

- Division durch Primzahl

Ein Ordnungsbegriffswert wird durch eine Primzahl dividiert. Der entstehende Rest dient als Adresse. Zur Division wird die Primzahl ausgewählt, die der Zahl der zu speichernden Sätze am nächsten ist, weil dann eine gute Streuung der Reste und damit wenig Mehrfachbelegungen zu erwarten sind.

- Basistransformation

Da sich bei vielen Ordnungsbegriffen die Werte in bestimmten Bereichen häufen, was eine ungleichmäßige Belegung ergäbe, transformiert man Schlüssel in ein anderes Zahlensystem mit einer höheren Basis, z.B. zur Basis 11 oder zur Basis 13. Die Ergebniszahl oder Teile daraus ergeben die Adresse.

Von all diesen Umrechnungsverfahren kann man keines generell als bestes Verfahren bezeichnen. Die Entscheidung ist von Fall zu Fall zu treffen. Manchmal erweist sich eine Kombination mehrerer Rechenmethoden als vorteilhaft. Für den einzelnen Fall muß eine günstige Formel ausgewählt werden.

Will man ein Verfahren bewerten, sollte man folgende Faktoren beachten:

- Speicherbelegung
- Zahl der Überlaufsätze
- Aufwand zur Errechnung der Adressen (so ist z.B. Falten leichter als Stellenauswahl.)

Trotz Überlaufsätze ergibt die indirekte Adressierung gute Werte für die mittleren Zugriffszahlen, die bei Verwendung eines gut ausgewählten Algorithmus praktisch immer kleiner als 2 bleiben.

Hauptnachteil gegenüber der direkten Adressierung ist aber nicht der schlechtere Zugriffszahlwert, sondern die Tatsache, daß die Sätze gestreut gespeichert sind und bei großem  $n$  eine erhebliche Bewegung des Zugriffsarms nötig wird.

### 3.2.2 Suchverfahren mit Zeigern (Pointern)

Bei der Anwendung von Suchverfahren mit Zeigern handelt es sich meist um die Bereitstellung von Daten zu Verarbeitungen ohne Angabe eines Ordnungsbegriffs. Die Sätze, die einem Suchargument entsprechen, können daher in ihrer Anzahl schwanken. Sucht man nach einem Ordnungsbegriff, so gibt es für jeden Satz nur einen Ordnungsbegriff (z.B. laufende Nummer). Durch einen Feldinhalt wird ein Satz nicht eindeutig bestimmt. Wird z.B. ein Feld "Familienstand" als Suchargument angegeben, so fallen viele Sätze aus, die einen gleichen Feldinhalt, z.B. ledig, haben. Wieviele Sätze bei einer Suche nach einem Feldinhalt ausfallen, hängt ab von den Anzahlen der Ausprägungen, die überhaupt möglich sein können.

Für Suchverfahren mit Pointern kann man in der Regel keine mittleren Zugriffszahlen ableiten. Vielmehr zieht man zur Bewertung die Zahl der Zugriffe heran, die benötigt werden, um zunächst den ersten der Suchbedingung entsprechenden Satz zu finden.

- a)     Adressverkettung
- 1)     Einfache Verkettung

Um logisch zusammengehörige Daten, die voneinander getrennt (physikalisch) gespeichert sind, zu verbinden, wendet man die Adressverkettung an. Zusammen mit einem Satz ist ein Kettungsfeld gespeichert, das die Adresse des folgenden logisch dazugehörigen Satzes enthält. Jeder Zugriff zu einem Kettungsfeld ergibt die Adresse für den nächsten Satz, der wiederum ein Kettungsfeld enthält. Die Verweisadressen bilden eine Kette. Eine solche Kette läßt sich durch einen sogenannten linearen Zugriffsgraphen darstellen, wobei die Knoten jeweils Daten und ein Kettungsfeld enthalten und die Kanten die durch die Kettungsfelder ermöglichten Zugriffe darstellen. Jeder Knoten, mit Ausnahme des ersten und des letzten Knotens, hat genau einen Vorgänger und einen Nachfolger. Der erste Knoten einer Adresskette wird als "Anker" bezeichnet, seine Adresse als "Ankeradresse". Der letzte Knoten enthält in der Regel keine weitere Adresse, sondern eine Kennzeichnung als Endekennzeichnung. Er kann aber auch die Adresse des ersten Knotens (des ersten Datensatzes) enthalten. In diesem Fall spricht man von zyklischer Geschlossenheit (AHRENS & WALTER, 1971).

- 2)     Doppelte Verkettung

Neben der einfachen Verkettung ist auch eine sogenannte doppelte Adressverkettung möglich (KNUTH, 1968). Bei doppelter Adressverkettung enthalten die Knoten zwei Kettungsfelder; eines davon enthält die Adresse des logischen Nachfolgers, das andere die des Vorgängers. Logisch zusammengehörige Sätze können daher in beiden Richtungen aufgesucht werden.

Adressverkettung bezeichnet man auch als Listenorganisation, wobei eine Liste definiert wird als eine geordnete Menge von Einheiten, in der jede Einheit eine Angabe der nachfolgenden Einheit enthält.

Der Begriff der Adressverkettung ist allerdings vorzuziehen, weil er unmittelbar etwas über das verwendete Ordnungsprinzip aussagt. Eine Verkettung kann natürlich nicht nur zwischen Sätzen einer Datei stattfinden, sondern auch zwischen anderen datenorganisatorischen Einheiten, wie z.B. einem Feld oder einem Segment.

Hauptsächlich werden Verkettungen zwischen Sätzen mehrerer Dateien und zwischen Sätzen einer Datei vorgenommen. Bezieht sich z.B. ein Suchargument auf ein Feld, das in verschiedenen Dateien vorkommt, so lassen sich durch Verkettung leicht die zugehörigen Daten aus allen Dateien zu einem logischen Satz zusammenstellen.

Bei der Verkettung zwischen den Sätzen einer Datei unterscheidet man die Verkettung der Sätze auf- oder absteigend nach ihren Ordnungsbegriffswerten. Dieses Vorgehen ist dann sinnvoll, wenn Sätze gestreut gespeichert sind und dennoch sequentiell verarbeitet werden müssen. Hier verlängert sich zwar die Suchzeit, da sich der Zugriffsarm einer Platte nicht kontinuierlich fortbewegen kann, sondern hin und her bewegt werden muß, jedoch kann der Vorteil der gestreuten Speicherung gegenüber der sequentiellen Speicherung (kein Verschieben der Folgesätze bei Änderungen) genutzt werden.

Die Änderungen der Verkettungen in gespeicherten Daten können nochmals unterschieden werden in solche Än-

derungen, die einen Speicherplatz tatsächlich freigeben, und in solche, die einen Speicherplatz als nicht mehr zugriffsfähig bezeichnen. Im Prinzip lassen sich allerdings Änderungen ohne Verschieben der bereits gespeicherten Sätze nur durch Änderungen der Kettungsfelder durchführen (KNUTH, 1968).

### 3) Feldwertverkettung

Eine weitere Möglichkeit der Verkettung ist die Feldwertverkettung. Man verkettet Sätze nicht über den Ordnungsbegriff, sondern über Felder eines Satzes. Diese Art der Verkettung ermöglicht die Verarbeitung von Daten ohne Angabe eines Ordnungsbegriffes in der Folge der Inhalte des verketteten Feldes. Hierbei können dann Voll- oder Teilketten unterschieden werden. Es kann dann die Datei sequentiell abgearbeitet werden, obwohl sie physikalisch in einer anderen Folge gespeichert ist.

Dieses Vorgehen ist zu empfehlen, wenn das Feld viele unterschiedliche Inhalte haben kann. Eine Teilkette verbindet nur Sätze, die für ein bestimmtes Feld denselben Inhalt enthalten. Solche Teilketten sind dann sinnvoll, wenn für die Inhalte eines Feldes nur wenige Ausprägungen vorhanden sind. Jede Teilkette enthält nur einen Teil der Sätze der Datei. Alle Teilketten für ein Feld umfassen alle Sätze.

### 4) Mehrfachverkettung

Werden die Sätze einer Datei für viele Felder miteinander verkettet (LEFKOWITZ, 1969), so entstehen zwar

sehr gute Zugriffsmöglichkeiten zu dieser Datei für Suchanfragen ohne Angabe des Ordnungsbegriffes; der sogenannte Pflegeaufwand für diese Datei steigt allerdings stark an, da beim Einfügen oder Löschen eines Satzes alle Kettungsfelder für Voll- oder Teilketten mitgeändert werden müssen. Außerdem müssen ja zunächst die Anker aufgefunden werden, ehe mit Hilfe der Kette auf alle benötigten Sätze zugegriffen werden kann.

Dieses Problem ist generell bei der Adressverkettung vorhanden. Deshalb werden häufig besondere Zugriffsmöglichkeiten zu den Ankern geschaffen, indem man Zeiger außerhalb der eigentlichen Datei speichert, die auf die Anker verweisen. So kann man bei einer über den Ordnungsbegriff verketteten Datei einen Zeiger angeben, der mitteilt, wo der Satz mit dem niedrigsten Ordnungsbegriffswert steht. Werden viele Adressketten für eine Datei angelegt, so kann man entsprechend alle Ankeradressen in einer kleinen Datei zusammenfassen, in einer sogenannten Ankeradressdatei (HSIAO & HARARY, 1970). Diese Datei enthält bei sogenannten Vollketten den Namen der verketteten Felder und die Adresse des ersten Satzes der Kette, bei Teilketten zusätzlich den Feldinhalt, für den die Teilkette angelegt worden ist.

Bei der Suche unterscheidet man daher auch das Aufsuchen der Ankeradresse vom Abarbeiten der Kette. Nur in dem zweiten Teil der Suche liegt eine Inhaltswiedergewinnung vor; hier wird für jeden der Suchbedingung entsprechenden Satz nur noch ein Zugriff benötigt. Zuvor muß mit einem Suchverfahren in der Ankeradressdatei die Ankeradresse ermittelt werden. Dies ist allerdings in der Regel kein Problem, da solche Dateien nur geringen Umfang haben.

Der Aufbau von Ankeradressdateien stellt die Vorstufe von Indizierungen (s.u.) dar.

## 5) Hierarchische Verkettung

Eine weitere Form der Verkettung zwischen Sätzen einer Datei liegt dann vor, wenn Sätze einer Datei logisch über- oder untergeordnet sind. Die Kettungsfelder verweisen in diesem Fall auf logisch übergeordnete oder untergeordnete Sätze. Wendet man auch hier die doppelte Verkettung an, die jeweils von einem beliebigen Satz ausgehen kann, so kann man dadurch komplexe logische Sätze entstehen lassen. Diese Sätze können hierarchische oder Baumstruktur haben.

Bei komplexen Strukturen müssen allerdings sehr viele Kettungsfelder für jeden Satz gespeichert werden. Man kann diese dann in einer eigenen Datei zusammenfassen. Die Ermittlung der logischen Strukturen kann dann in dieser kleineren und deshalb schneller abzuarbeitenden Datei erfolgen.

Auch bei diesem Verfahren ist nur eine Darstellung linearer Strukturen zwischen Sätzen möglich, was sich dadurch ausdrückt, daß für jedes verkettete Feld immer nur auf einen Nachfolger verwiesen werden kann.

Diese Verfahren fallen unter die nichtlineare Datenorganisation.

b) Indizierung

Im Gegensatz zur Adressverkettung werden die Pointer nicht in den Datensätzen gespeichert, sondern in speziellen Dateien, in den sogenannten Indextabellen. Für jedes Feld einer Datei, dessen Wert als Suchargument verwertet werden soll, wird eine Indextabelle angelegt. Jeder Satz dieser Tabelle enthält einen Inhalt des indizierten Feldes und die Adressen der indizierten Datei, die diesen Feldinhalt aufweisen. Es ist somit nur ein Suchvorgang zum Auffinden der richtigen Indexzeile notwendig, und ein weiterer für jeden benötigten Satz.

Eine Indextabelle kann als eine Datei mit linearen Sätzen aufgefaßt werden. Sie kann daher sequentiell oder gestreut gespeichert werden, und es ist auch möglich, ein beliebiges Suchverfahren in dieser Indextabelle anzuwenden (Suchverfahren ohne Zeiger). In Indextabellen zu suchen, ist schnell zu bewältigen, da diese meist sehr klein sind. Die Suche selbst entspricht einer Suche in einer Datei mit Ankeradressen.

Beim Abarbeiten einer Adresskette wird für jeden durch das Suchargument definierten Satz nur noch ein Zugriff benötigt. Man kann aber bei der Indizierung die Adressen der Sätze aufsteigend sortieren und fortlaufend auf das Speichermedium zugreifen. Befinden sich mehrere gesuchte Elemente, die auch physikalisch hintereinander gespeichert worden sind, in einem Block, so kann das zu erheblichen Zeiteinsparungen führen.

1) Primärindizierung

Bei der Primärindizierung wird für den Ordnungsbegriff einer Datei im Primärindex eine Indextabelle angelegt (LUTZ & KLIMESCH, 1971). Dieses Verfahren ist nur sinnvoll, wenn es sich um kleine Indextabellen handelt und die Suche im Kernspeicher durchgeführt werden kann. Sie wird verwendet, wenn sequentiell organisierte Daten vorhanden sind, um kurze Suchzeiten zu erhalten (DUDECK, 1972).

Im Rahmen von indexsequentiellen Zugriffsmethoden ist dies die gebräuchlichste Form. Das Suchen in einer Indextabelle geschieht dann, indem immer nur ein Block, der gerade durchgesucht wird, in den Kernspeicher geholt wird.

Primärindizierung ist besonders dann interessant, wenn die Ordnungsbegriffswerte einer Datei für die direkte Adressierung benutzt werden können, die Sätze aber unterschiedliche Länge haben, d.h. es läßt sich eine Indextabelle für die Datei der Primärindizes aufbauen, mit der eine direkte Adressierung möglich ist. Die indizierte Datei selbst kann beliebig gespeichert werden, z.B. sequentiell oder gestreut.

2) Sekundärindizierung (Invertierung)

Diese Form ist die wichtigste Anwendung der Indizierung. Hier werden Indextabellen für Nicht-Ordnungsbegriffswerte angelegt. Hiermit kann ein schneller Zugriff zu Daten erfolgen, die zur Verarbeitung ohne Angabe eines Ordnungsbegriffswertes benötigt werden.

Bei der Primärindizierung enthält jede Indexzeile nur eine Adresse, da jeder Satz der indizierten Datei einen anderen Ordnungsbegriffswert hat. Bei der Sekundärindizierung können dagegen mehrere Adressen in einer Indexzeile auftreten. Die Indextabellen sind meist aufsteigend nach Feldinhalten sortiert. Die Primär- und Sekundärindizierung wird mit inverted file (invertierte Datei) bezeichnet, im Gegensatz zum direct file oder usual file (vgl. RS-II).

Wenn Indextabellen für alle Felder einer Datei angelegt werden, so ist die Datei voll invertiert, d.h. man benötigt 100 % mehr Speicherplatz. Meist jedoch ist der Aufbau von Sekundärindizes nur für die Felder, die häufig in Suchanfragen benötigt werden, sinnvoll (CARDENAS, 1975).

### 3) Mehrdimensionale Indizierung

Hier werden in einer Indextabelle Adressen aus mehreren Dateien gespeichert. Enthalten mehrere Dateien dasselbe Feld, so kann mit der Suche in der Indextabelle festgestellt werden, in welchen Sätzen der Dateien ein bestimmter Wert dieses Feldes auftritt. So lassen sich logische Sätze aus den physischen Sätzen mehrerer Dateien zusammenstellen.

Während bei der Sekundärindizierung mehrere Indextabellen für eine Datei bestehen, verweist ein mehrdimensionaler Index auf mehrere Dateien.

#### 4) Binärmatrizen

Bei der Sekundärindizierung ist bei sehr vielen indizierten Feldern eine große Anzahl von Indextabellen zu speichern. Man kann bei Feldelementen, die nur wenige Ausprägungen haben, die Technik der Binärmatrix verwenden. Dabei werden in jeder Indexzeile nicht die Adressen der Sätze gespeichert, sondern es wird nur die relative Stellung der Sätze in der indizierten Datei vermerkt. In jeder Indexzeile wird für jeden gespeicherten Satz der indizierten Datei ein Bit vorgesehen. Das erste Bit entspricht dem ersten Satz, u.s.f.

Enthält ein gespeicherter Satz in der Indexzeile den angegebenen Wert, so hat das entsprechende Bit den Wert 1, sonst 0. Wird eine invertierte Datei nach diesem Prinzip aufgebaut, so entsteht eine Binärmatrix, wobei jede Zeile einem Feldinhalt und jede Spalte einem gespeicherten Satz entspricht. Da die Spaltenbeschreibung konstant ist, können mehrere Indextabellen in einer Matrix gespeichert werden, wenn man die Zeilenbeschreibungen durch die Feldnamen ergänzt. Die Zeilen einer Matrix werden Bit-Ketten genannt.

Werden einmal gespeicherte Sätze einer invertierten Datei nicht geändert und neue Sätze seriell am Ende der Datei angefügt, so beschränkt sich das Ändern der Binärmatrix auf das Anfügen neuer Spalten. Wird eine Datei voll invertiert und werden die Indextabellen mit Hilfe von Binärmatrizen gespeichert, so kann man auf eine Speicherung der direkten Daten verzichten, da sämtliche mögliche Dateninhalte der Sätze in den Zeilen der Matrix enthalten sind.

Diese Form der Invertierung ist aber nicht immer optimal, wie wir weiter unten zeigen werden (bei der Anwendung von RS-II).

### 3.3 Such- und Änderungsverfahren mit Bäumen (Nicht-lineare Listen)

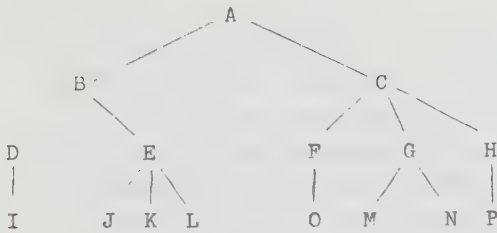
Eine Baumstruktur kann auf verschiedene Weisen, die einander äquivalent sind, dargestellt werden.

Die gebräuchlichsten Darstellungen sind die Blockdarstellung, die geschachtelte Klammerdarstellung oder die Graphen(Baum)-Darstellung (WIRTH, 1975).

```
A.  
B  
  D  
    I  
  E  
    J  
    K  
    L  
C   Blockdarstellung (durch Einrücken)  
  F  
    O  
  G  
    M  
    N  
  H  
    P
```

(A(B(D(I),E(J,K,L)),C(F(O)G(M,N),H(P))))

geschachtelte Klammerdarstellung



Graphendarstellung

Die Graphendarstellung hat auch zur Bezeichnung Baum geführt.

Wir unterscheiden an Bäumen Knoten und Kanten. Der oberste Knoten wird als Wurzel oder Kopf bezeichnet. Die letzten (unteren) Knoten sind die Blätter oder Endelemente. Die Zahl der unmittelbaren Nachfolger eines Knotens bezeichnet man als Grad dieses (inneren) Knotens. Der höchste vorkommende Grad ist der Grad des Baumes. Ein Knoten, der direkt einem anderen untergeordnet ist, wird als Nachfolger oder Sohn bezeichnet, einer, der einem Knoten direkt übergeordnet ist, hat die Bezeichnung Vorgänger oder Vater. Knoten befinden sich auf sog. Stufen. Die Wurzel hat definitionsgemäß die Stufe 1. Die größte Stufe eines Baumes ist seine Tiefe, bzw. Höhe.

Sind die Nachfolger jedes Knotens geordnet (d.h. kommt es auf ihre Reihenfolge an), so hat man einen übergeordneten Baum vor sich (z.B. die Sonderfälle, AVL-Bäume bzw. B-Bäume s.u.).

### 3.3.1 Binärbäume

Allgemeine Binärbäume haben für jeden Knoten 0,1 oder 2 Nachfolger. Sie sind geordnete Bäume von Grad 2. Gilt für einen Binärbaum für jeden Knoten, daß alle Schlüssel im linken Unterbaum (Teilbaum) kleiner sind als der Schlüssel des Knotens, und gilt ferner, daß alle Schlüssel im rechten Teilbaum größer sind als der Schlüssel des Knotens, so ist dieser Binärbaum ein Suchbaum (search tree).

Ein Baum ist vollständig (völlig) ausgeglichen (ausgewogen), wenn sich für jeden Knoten die Anzahl der Knoten im rechten Unterbaum von der Anzahl der Knoten im linken Unterbaum um höchstens 1 unterscheidet.

Eine schwächere Definition der Ausgeglichenheit (unvollständige Ausgeglichenheit, AVL-Ausgeglichenheit s.u.) ist die folgende:

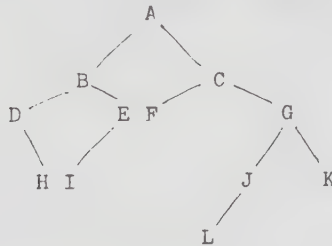
Ein Baum ist genau dann (AVL-) ausgeglichen, wenn sich für jeden Knoten die Höhen der Unterbäume um höchstens 1 unterscheiden. AVL-Bäume werden unten als Sonderfall von Binärbäumen behandelt. Vollständig ausgeglichene Bäume sind auch (AVL-) ausgeglichen.

#### a) Verarbeitungsformen an Binärbäumen

Will man alle Knoten eines Baumes aufsuchen, so muß man wissen, wie die Knoten dieses Baumes gespeichert sind, d.h. in welcher linearen Ordnung. Mit der Speicherungsform ist auch die Reihenfolge des Durchlaufens festgelegt. Es werden die drei Formen unterschieden:

- Vorordnung (preorder)  
Speichern der Wurzel, des linken und des rechten Nachfolgers
- Zwischenordnung (postorder)  
Speichern des linken Nachfolgers, der Wurzel und des rechten Nachfolgers
- Nachordnung (enorder)  
Speichern des linken Nachfolgers, des rechten Nachfolgers und der Wurzel

Gegeben sei z.B. der folgende Binärbaum:



Dann werden die einzelnen Knoten (=logische Sätze) bei der Vorordnung in der folgenden Reihenfolge gespeichert, bzw. aufgesucht (NIEMEYER, 1975).

A B D H E I C F G J L K

bei der Zwischenordnung in der Reihenfolge

D H B I E A F C L J G K

und bei der Nachordnung

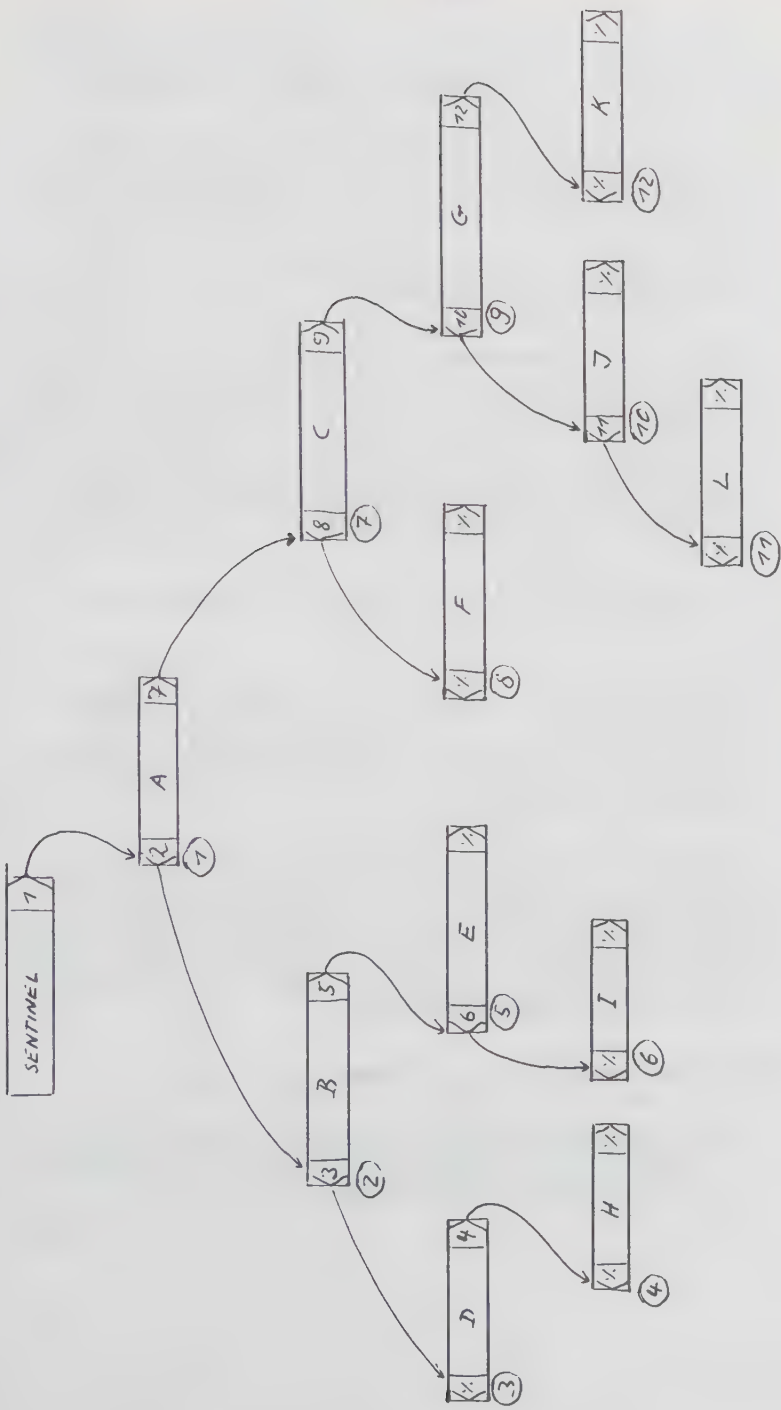
H D I E B F L J K G C A

Für jeden Knoten wird demnach sofort wieder das betreffende Ordnungsprinzip angewendet. (=Rekursive Prozeduren, die zweckmäßigerweise als rekursive Operatoren definiert werden).

Eine optimale Speicherungsart für einen Binärbaum, der einmal aufgebaut wird und dann unverändert bleibt, ist die einfach gekettete Speicherung. Dabei weist jeder Knoten, außer dem Wurzelknoten, zwei Pointer auf, jeweils einen zum linken Nachfolger und einen zum rechten Nachfolger (bzw. Unterbaum). Zeiger können auch leer sein (Endesymbol).

Die Knoten (=logische Sätze) sind in der Verordnung (pre-order) auf dem Speichermedium untergebracht.

Einfach gekettete Speicherung in Vorordnung



Beim Einfügen von Knoten in einen Baum muß zunächst die Stelle, an der eingefügt werden soll, identifiziert werden. Erst dann kann ein Knoten eingefügt werden. Allerdings dürfen nur Endknoten oder Knoten mit einem Unterbaum eingefügt werden. Außerdem muß der Unterbaum des einzufügenden Knotens dieselbe Richtung haben wie der einzufügende Knoten in Bezug auf seinen Wurzelknoten.

Die gleiche Restriktion gilt auch für das Entfernen von Knoten. Es dürfen nur solche Knoten entfernt werden, die entweder Endknoten sind, oder solche, die nur einen Unterbaum haben, der sich in der gleichen Richtung befindet wie der zu löschende Knoten in Bezug auf seinen Wurzelknoten.

Bei allen anderen Fällen wäre eine vollständige Reorganisation des Baumes notwendig.

Soll während einer Anwendung die Struktur des Baumes verändert werden, d.h. soll der Baum dynamisch wachsen und schrumpfen können, so sind andere Organisationsformen notwendig.

Eine dieser Formen ist die Rückverkettung (Rückwärtsverzeigerung). Statt Pointer auf Null (allgemeines Endesymbol) zu setzen, wenn von diesem Knoten kein Unterbaum entspringt, läßt man die Pointer auf hierarchisch höhere Knoten verweisen.

Es gelten folgende Regeln (NIEMEYER, 1975):

- Eine Rückverkettung führt von einem rechten Pointer zu dem Vorgänger in Zwischenordnung.

- Eine Rückverkettung führt von einem linken Pointer zu dem Vorgänger in Zwischenordnung.
- Eine Rückverkettung führt jeweils vom äußersten linken und vom äußersten rechten Knoten einer Zwischenordnung zum sentinel (Kopf, Wurzel).

Für einen derart organisierten Baum hat KNUTH 1973 Algorithmen für das Durchsuchen (in Zwischenordnung), Einfügen und Löschen von Knoten angegeben. Werden die Rückwärtspointer statt auf die Zwischenordnungsvorgänge bzw. Nachfolger auf die höchsten sich in gerader Linie befindlichen Vaterknoten geführt, so entsteht eine zyklisch verkettete Baumorganisation. Diese ist vor allem für die Binärbaumdarstellung von Mehrwegbäumen von Bedeutung.

b) AVL-Bäume  
(ADELSON-VELSKI, LANDIS, 1975)

AVL-Bäume sind Sonderfälle von Binärbäumen, die besonderen Bedingungen genügen. Für jeden Knoten gilt, daß der Schlüsselwert des rechten Nachfolgers größer ist als der Schlüssel des Knotens und daß der Schlüsselwert des linken Knotens kleiner ist als der Schlüsselwert des Knotens. Außerdem darf sich die Länge des längsten Weges im linken Unterbaum von der des rechten Unterbaums für jeden Knoten um höchstens 1 unterscheiden.

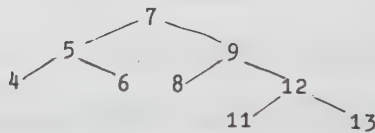
Man unterscheidet danach links- und rechtslastige, sowie gleichgewichtige Knoten. Bei Änderungsvorgängen (Hinzufügen von Knoten) können folgende Situationen auftreten (neue Knoten werden immer als Endknoten angefügt):

- Ein gleichgewichtiger Knoten wird links- oder rechtslastig.
- Ein links- oder rechtslastiger Knoten wird gleichgewichtig.
- Ein neuer Knoten soll an einen links- oder rechtslastigen Knoten angehängt werden. Die AVL-Bedingung wird verletzt, dieser Knoten heißt dann kritischer Knoten. In diesem Fall muß eine Reorganisation des Baumes vorgenommen werden, um die AVL-Eigenschaft wieder herzustellen.

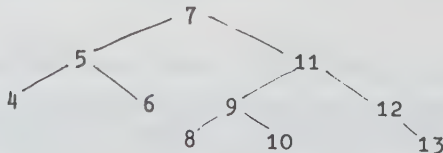
Hierfür gibt es Algorithmen (vgl. KNUTH, 1973).

Beispiel:

Gegeben sei der folgende AVL-Baum



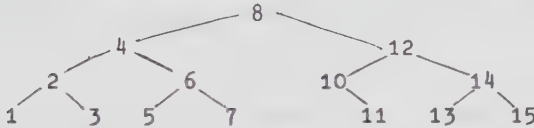
Es soll ein Knoten mit dem Schlüsselwert 10 eingefügt werden. Dieser müßte links an 11 angefügt werden. Damit wäre 11 ein kritischer Knoten. Eine Reorganisation des Baumes ergäbe folgendes Bild:



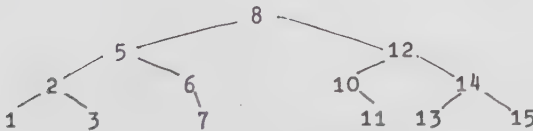
Beim Löschen von Knoten muß an die Stelle des gelöschten Knotens ein Endknoten hochgezogen werden. Dabei muß die Entscheidung getroffen werden, ob dies der kleinste größere Schlüssel oder der größte kleinere Schlüssel sein soll.

Beispiel:

Gegeben sei der folgende AVL-Baum



Es soll der Knoten mit dem Schlüssel 4 gelöscht werden. Es ergibt sich dabei dann folgendes Bild:



In vielen Fällen zeigt sich, daß das erste Entfernen eines Knotens eine Reihe von Veränderungen nach sich zieht.

c) Binärbaumdarstellung von allgemeinen Mehrwegbäumen

Um einen allgemeinen Baum als Binärbaum darzustellen, muß jeder Vaterknoten mit seinem äußersten linken (ersten) Sohnknoten verknüpft werden. Außerdem müssen alle Sohnknoten untereinander einfach verkettet werden.

Dadurch wird vermieden, daß in einem Vaterknoten eine variable Anzahl von Pointern untergebracht werden müssen. Die Binärbaumdarstellung von Mehrwegbäumen hat heute nur noch geringe Bedeutung.

### 3.3.2 Mehrwegbäume

Ein Anwendungsgebiet für Mehrwegbäume ist die Konstruktion von großen Suchbäumen, die sich dynamisch verändern.

Für gewöhnlich werden diese Bäume in Teilbäume (Seiten = pages) unterteilt, diese werden dann in einer zugreifbaren Speichereinheit untergebracht. Damit muß nicht auf jeden Knoten einzeln zugegriffen werden.

Bei den Mehrwegbäumen sind zwei Schwierigkeiten zu überwinden:

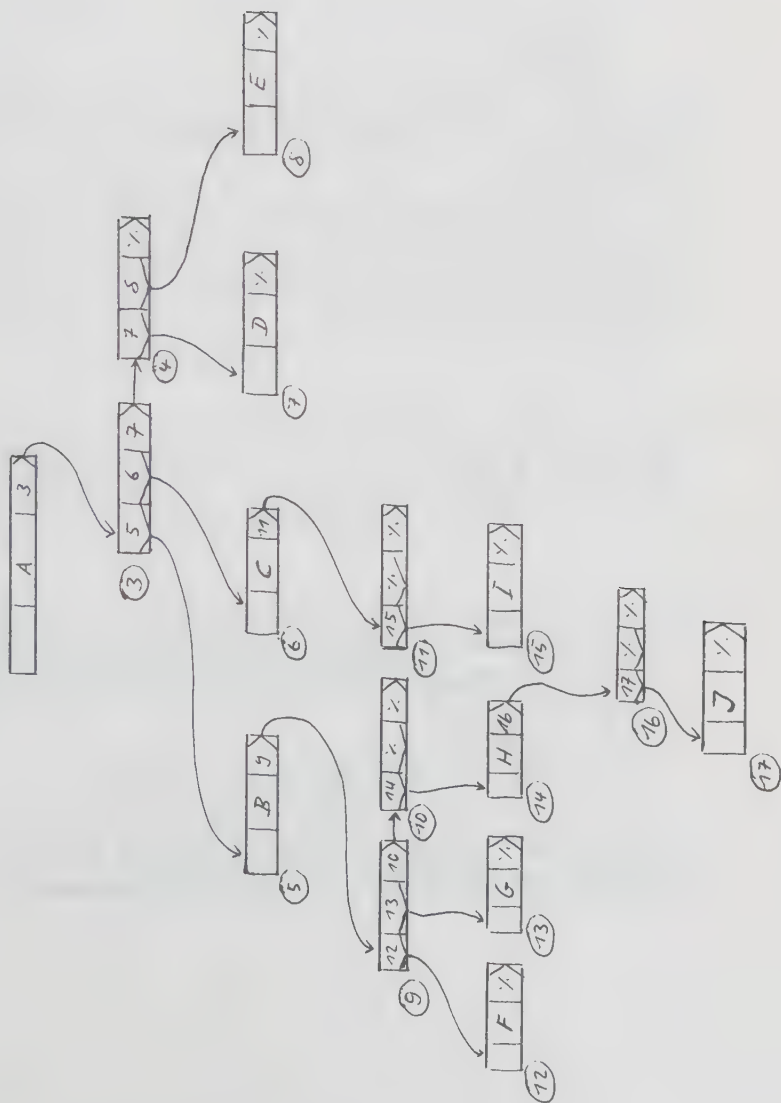
- die variable Anzahl von Pointern in einem Knoten
- das Wachstum des Mehrwegbaumes unter Kontrolle zu halten

#### a) Darstellung von Mehrwegbäumen mit Indexknoten

Eine interessante Lösung für dieses Problem schlägt NOLTEMEYER 1972 vor. Hier wird vermieden, daß unregelmäßige Baumstrukturen (variable Formate) auftreten. Außerdem braucht in einem Knoten nicht eine Maximalanzahl von Pointern vorgesehen werden.

Von jedem Vaterknoten zeigt ein Pointer auf einen logischen Satz (Indexknoten), der die Adressen (Pointer auf) und Schlüssel sämtlicher Sohnknoten enthält, aber keine

Darstellung eines Mehrwegbaumes mit Hilfe von "Indexknoten"



Daten. Sind viele Sohnknoten vorhanden, so können auf der gleichen Ebene mehrere Indexknoten untereinander wiederum einfach verkettet werden. Die Abbildung (S.65) zeigt einen Teil eines Baumes, der auf diese Weise organisiert ist.

b) B-Bäume

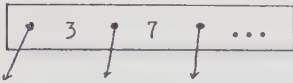
Der wichtigste Aspekt der B-Bäume ist die Kontrolle des Wachstums von Mehrweg-Suchbäumen (BAYER & MCCRIGHT, 1972; KNUTH, 1973).

Mit Hilfe von B-Bäumen (B-Trees) können größere dynamische Indexmengen (Schlüsselmengen) verwaltet und verarbeitet werden. Ein Knoten eines B-Baumes sieht wie folgt aus:



wobei  $P_1$  = Zeiger (Pointer) auf Nachfolger-Knoten (Söhne)  
 mit den Schlüsseln größer als  $K_1$  aber kleiner  
 als  $K_{1+1}$   
 $K$  = Schlüssel (key)  
 $V$  = (Daten-) Wert oder Zeiger auf Daten

In vereinfachter Darstellung werden nur die Schlüssel als Zahlen und die Zeiger als Pfeile wiedergegeben.



Es gelten weiterhin folgende Konventionen:

- Der Wurzelknoten kann zwischen 1 und  $2k$  Schlüssel enthalten (damit zwischen  $2$  und  $2k+1$  Nachfolger haben).
- Jeder andere Knoten enthält zwischen  $k$  und  $2k$  Schlüssel (damit zwischen  $k+1$  und  $2k+1$  Nachfolger). Nur die Endknoten haben keine Nachfolger.
- Jeder Weg vom Wurzelknoten zu einem Endknoten hat die gleiche Länge (Höhe  $h$ ), d.h. alle Endknoten liegen auf der gleichen Stufe.

Die Ordnung  $n$  eines B-Baums wird in der Literatur unterschiedlich definiert:

entweder  $n = 2k$  (obige Darstellung)  
oder  $n = k$ .

Für das Aufsuchen und Eintragen von Schlüsseln, sowie das Löschen von Schlüsseln in B-Bäumen gibt es Algorithmen, die im folgenden im Prinzip beschrieben werden sollen:

Das Aufsuchen von Schlüsseln geschieht im Zuge des Durchwanderns eines Baumes. Beginnend mit dem Wurzelknoten wird ein Knoten (linear) durchsucht. Bei Gleichheit ist der Schlüssel gefunden, das hinter dem Schlüs-

sel stehende Datum kann verarbeitet werden (da, wie oben erwähnt, dies evtl. auch ein Pointer auf einen Datenbereich sein kann, ist es möglich mit diesen Daten zu arbeiten).

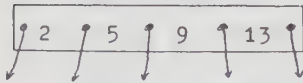
Ist der zu suchende Schlüssel kleiner als der erste Schlüssel in einem Knoten, so wird mit der Suche in dem Knoten fortgefahren, auf den der Zeiger weist. Ist beim linearen Durchsuchen der gesuchte Schlüssel zum erstenmal größer als ein Schlüssel im Knoten, so wird mit der Suche in dem Knoten fortgefahren, auf den der Zeiger  $pi$  weist.

Tritt der Fall ein, daß in einem Endknoten noch keine Gleichheit erreicht wird, so ist der gesuchte Schlüssel nicht vorhanden (es wurde ein leerer Pointer (Nullpointer) erreicht).

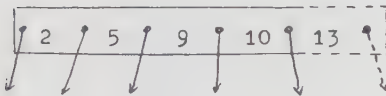
Das Eintragen eines neuen Schlüssels geht wie folgt vor sich:

Bei der Suche wird ein Endknoten (leerer Pointer) erreicht. Sind im Endknoten weniger als 2  $k$  Schlüssel vorhanden, so wird der neue Schlüssel eingeordnet und ein neuer leerer Pointer vorgesehen. Sind im Endknoten bereits 2  $k$  Schlüssel vorhanden, wo wird dieser Knoten aufgeteilt (gesplittet): Der neue Schlüssel wird eingeordnet und ein neuer leerer Pointer geschaffen. Es entsteht ein Überlauf. Der Knoten wird geteilt, indem der nunmehr mittlere Schlüssel herausgenommen, d.h. nach oben weitergegeben wird, der verbleibende 'linke' Teil bleibt im Knoten, der 'rechte' Teil bildet einen neuen Knoten.

Beispiel: Gegeben sei ein Knoten der folgenden Form:



Es soll der Schlüssel 10 eingefügt werden.



Der Knoten wird gesplittet, der nunmehr mittlere Schlüssel km nach oben gegeben.

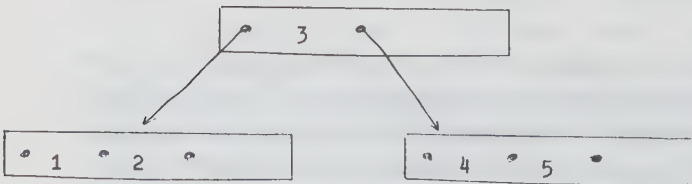
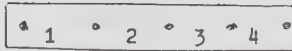
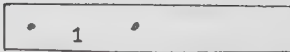


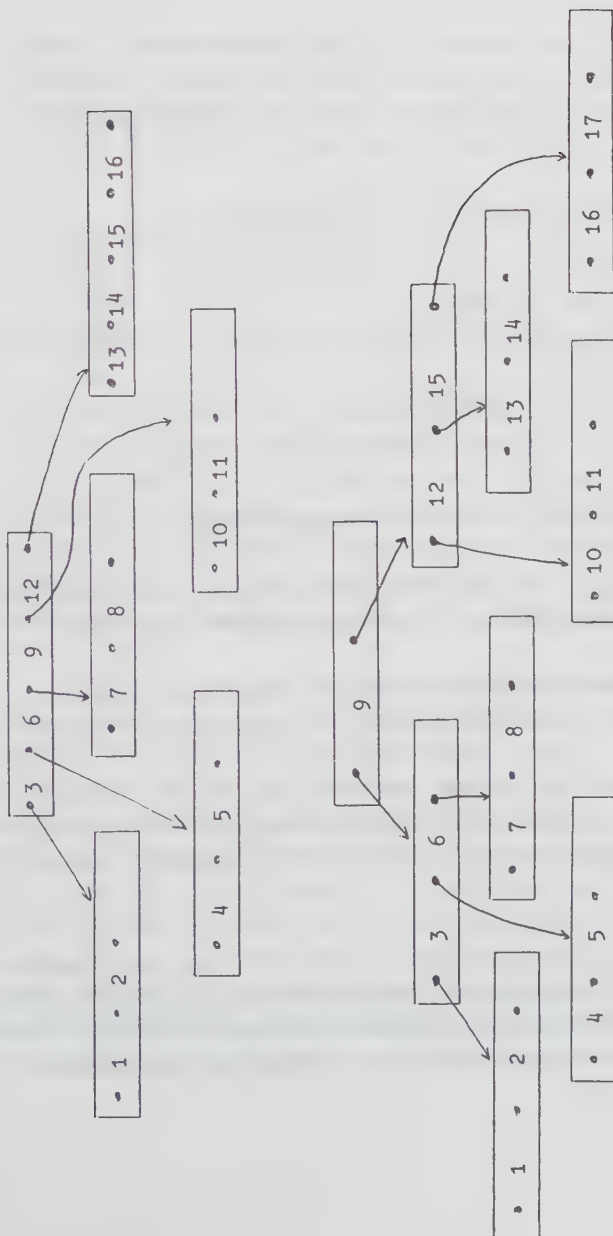
Der mittlere Schlüssel wird nach dem gleichen Verfahren in den Knoten der nächsthöheren Stufe eingeordnet, wobei ein Pointer pm geschaffen werden muß, der mitgegeben wird und auf den neu entstandenen Knoten ('rechten' Teil) verweist. Dieses Vorgehen kann sich bis zum Wurzelknoten hin fortsetzen.

Enthält der Wurzelknoten  $2k$  Schlüssel, und soll ein Schlüssel, der von unten hochgegeben wird, eingefügt werden, so wird der Wurzelknoten, wie oben beschrieben,

geteilt. Der nach oben gegebene mittlere Schlüssel bildet den neuen Wurzelknoten, die Pointer  $p_0$  und  $p_1$  verweisen auf die beiden Teile des ursprünglichen Wurzelknotens.

Beispiel: Es soll ein B-Baum der Ordnung  $m = 2$  mit den Schlüsseln 1-17 aufgebaut werden; die folgende Abb. zeigt einige Stufen:





Nach Einfügen des Schlüssels 17 hat der B-Baum die Höhe 3 erreicht.

B-Bäume sind deshalb vorteilhaft anzuwenden, weil die Effizienz der Operationen des Suchens, Eintragens und Löschens gleich sind. Sie sind proportional  $\log_k E$  ( $E$  = Anzahl der Eintragungen).

Die minimale Anzahl der Eintragungen ist

$$E_{\min} = 2(k+1)^{h-1} - 1$$

und die maximale Anzahl

$$E_{\max} = (2k+1)^h - 1.$$

Grenzen für  $h$  sind somit

$$\log_{2k+1} (E+1) \leq h \leq 1 + \log_{k+1} \frac{E+1}{2}.$$

Die mittlere Speicherausnutzung beträgt 75 %.

Das Löschen von Schlüsseln in B-Bäumen gestaltet sich etwas komplizierter. Wir unterscheiden die Fälle:

- Löschen eines Schlüssels in einem Endknoten
- Löschen eines Schlüssels in einem anderen Knoten

Besonders ist bei dem Löschvorgang auf das Eintreten eines sog. Unterlaufs (Underflows) zu achten, d.h. darauf, daß sich zuwenig Elemente in einem Knoten befinden.

Diesen Zustand kann man dadurch beheben, daß benachbarte Seiten angegliedert (Concatenation) werden. Entstehen durch diese Operation Knoten, die zuviele Elemente enthalten, muß wiederum ausgeglichen werden.

Ein besonderer Fall tritt ein, wenn der Wurzelknoten selbst verschwindet. In diesem, und nur in diesem, Fall wird der B-Baum in seiner Höhe kleiner. Die Algorithmen für das Löschen von Schlüsseln sind relativ umfangreich, weshalb in diesem Rahmen auf ihre genauere Beschreibung verzichtet werden soll. (WIRTH, 1975; KNUTH, 1973).

#### 4 Umfassende Datenbankmodelle

##### 4.1 CODASYL

###### 4.1.1 Bericht der CODASYL-Data-Base-Task-Group

Geht man mit umfangreichen Datenmengen um, die vielfachen Anwendungen unterworfen sind, so entsteht bald der Wunsch, soweit wie möglich von den physikalischen und organisatorischen Eigenschaften von Datenverarbeitungsanlagen zu abstrahieren und sich nur noch mit den statischen und dynamischen Eigenschaften einer vorgegebenen Problemstellung befassen zu können.

Erste Ansätze zur systematischen Beschreibung von Datenstrukturen kommen aus dem kommerziellen Bereich (Schaffung der Programmiersprache COBOL, 1959). Hier wird zwischen der Beschreibung der auszuführenden Operationen und der Beschreibung der davon betroffenen Datenobjekte klar unterschieden. Die Gesamtkonzeption ist jedoch auf sequentielle Speicherung, d.h. auf Datenträger wie Magnetband, ausgerichtet.

Als die Möglichkeit geboten wurde, direkte Zugriffsspeicher zu benutzen, wurde zunächst auf einer geräteorientierten und damit implizit problemfernen Stufe

mit Datenorganisations- und Datenverwaltungs-Systemen begonnen. Eine Arbeitsgruppe, die Conference on Data Systems Language, abgekürzt CODASYL, versuchte deshalb in mehrjähriger Arbeit, eine Sprache zu schaffen, die der Datenbeschreibung unter modernen Anforderungen gerecht wird. Diese Vorschläge wurden unter dem Namen Data-Base-Task-Group-Report (DBTG-Report, 1971) bekannt.

Bald war das Ziel der Arbeitsgruppe nicht mehr die Anpassung von COBOL an die modernen Bedürfnisse, sondern man versuchte, allgemeine und systematische Grundlagen zur Beschreibung von Datenstrukturen für den kommerziellen Bereich zu schaffen. Auf diese Weise wurde die Standardisierung von Benutzerschnittstellen von Datenbanksystemen beschleunigt; dennoch werden im wesentlichen erst in neuester Zeit die Gedanken der DBTG aufgenommen und mit Erfolg verwirklicht. Die DBTG-Vorschläge sind keineswegs unumstritten. Unabhängig von diesen Vorschlägen sind zu gleicher Zeit oder später weitere und abweichende Ansätze entwickelt worden (Streit zwischen Anhängern des DBTG-Vorschlags und Vertretern des Relationalmodells).

Es ist interessant, an dieser Stelle zu vermerken, daß ein zweites CODASYL-Komitee den DBTG-Bericht selbst wieder zum Gegenstand eigener Standardisierungsbemühungen gemacht hat.

Im allgemeinen geht die problemorientierte Darstellung von Datenstrukturen auf Kosten der Effizienz der Verarbeitung. Der DBTG-Bericht sagt daher von Anfang an, daß es nicht möglich sein wird, eine Datenbank im Hinblick auf ihre Verwendung in allen Einzelheiten zu optimieren. Es wird wie folgt argumentiert:

- a) Der Benutzer als Programmierer hat die Kontrolle über Speicherstrukturen immer mehr an zentrale Datenverwaltungsprogramme delegiert.
- b) Die Mehrfachverwendung derselben Daten von mehreren Benutzern für verschiedene Zwecke ist heute die Regel.
- c) Eine Mehrfachspeicherung (redundante Speicherung) ist bei dem Umfang der heutigen Datenbestände meist nicht mehr möglich. Man geht vielmehr von einer Grundform aus, die vor Gebrauch jeweils erst aufbereitet werden muß.
- d) Die Frage der Optimierung tritt mehr in den Hintergrund, da moderne Speicherungsmedien, die zudem in den Kosten extrem gesenkt wurden, zur Verfügung stehen.

Es wird daher von der Arbeitsgruppe gefordert, daß Datensammlungen heute so entwickelt werden, daß sie sich von verschiedenen Benutzern für verschiedene Zwecke auswerten lassen, und daß dies in verschiedenen Sprachen möglich ist. Folgende Möglichkeiten sollten hierzu geboten werden:

a) Beschreibung von Objekten und Typen

1. Wenn Objekte verschiedenartig benutzt werden, soll für diese Objekte auch möglich sein, verschiedenartige Beschreibungen je nach Anwendungsart einzusetzen, d.h. die Datenstrukturen für eine Anwendung lassen sich in einer bestimmten Darstellung beschreiben.
2. Die Beschreibung derselben Daten kann von Fall zu Fall unterschiedlich sein. Hinsichtlich der verwendeten Ebene der Beschreibung bestehen ebenfalls Unterschiede (Datenbasisverwaltung, Anwendungsebene).

3. Mit wenigen einfachen Darstellungsmitteln läßt sich eine Vielzahl verschiedenartiger Datenstrukturen beschreiben.
4. Die Beschreibung der Datensammlung läßt sich mit verschiedenen Sprachen kombinieren, in denen dann weitere Auswertungen beschrieben werden können.

b) Mehrfachzugriff zu den Daten

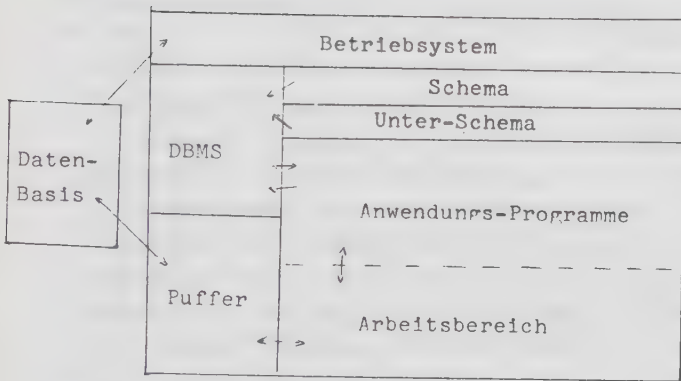
Mehrere Programme können gleichzeitig dieselbe Datenbasis auswerten. Dabei muß für jede Anwendung eine eigene Suchauswahlstrategie erlaubt sein. Weiter ist erforderlich, daß der Datenschutz für die Datenbasis oder Teile davon gegeben ist. Eine gegenseitige Beeinflussung der Programme muß ausgeschlossen sein.

c) Speicherstrukturen

1. Die Verwaltung der verfügbaren Speicher wird zentral vorgenommen.
2. Es existiert Programmunabhängigkeit von der physikalischen Konfiguration der EDV-Anlage und von der physikalischen Speicherrealisierung der Datenstrukturen innerhalb von Speichermedien.
3. Eine gewisse Form der Optimierung soll dadurch erreicht werden können, daß sich die gewählte Form für die Speicherstruktur nach dem überwiegenden Gebrauch der Datenstrukturen richtet.

4.1.2 Datenbasis-Verwaltungs-System

Die Systemvorstellung, die den DBTG-Vorschlägen zugrunde liegt, sei im folgenden skizziert.



Wir nehmen zunächst an, es existiere nur eine einzige Datenbasis, die auf einem Speichermedium angelegt sei (Hintergrundspeicher). Der Typ dieser Datenbasis wird durch ein sogenanntes Schema beschrieben, das sich im Hauptspeicher befindet. Die Grundoperationen innerhalb dieser Datenbasis wie Einfügen, Löschen, Ändern werden über ein Daten-Basis-Verwaltungs-System (abgekürzt DBMS) ausgeführt. Dieses benötigt u.a. das Schema. Die Übertragung der Daten zwischen Haupt- und Hintergrundspeicher geschieht über das Betriebssystem.

Im folgenden sei das Schema näher erläutert. Es beschreibt den Typ der Datenbasis mit Hilfe von einheitlichen und anwendungsunabhängigen Begriffen. Für bestimmte Grundformen, die übergreifend für verschiedene anwendungsspezifische Probleme sind, nennen wir diese Beschreibung, die die Speicherstruktur beschreibt, bzw. aus der sich die Speicherstruktur ableiten läßt, ein Schema.

Ist die Beschreibung vollkommen anwendungsspezifisch für eine Anwendung, so spricht man von einem Unterschema (Subschema).

Jedes Schema ist in einer Datenbeschreibungssprache (Data-Description-Language, DDL) abgefaßt. Da sich Schema und Unterschema nicht völlig entsprechen müssen, ist während der Datenübertragung eine Pufferfunktion notwendig. Da mehrere Anwenderprogramme mit denselben oder unterschiedlichen Unterschemata gleichzeitig von der Datenbasis Gebrauch machen können, muß jedem einzelnen Benutzer ein Arbeitsbereich zugeteilt werden.

Die Programme können dann auch in unterschiedlichen Sprachen abgefaßt sein. Das Datenbasis-Verwaltungs-System ruft sie jedoch über eine einheitliche Schnittstelle auf, die Daten-Manipulations-Sprache (Data-Manipulation-Language, DML). Diese ist jeweils in eine Programmiersprache eingebettet und stellt eine sogenannte Host-Language (Gast-Sprache) dar.

Welche Schritte im einzelnen bei der Anforderung bzw. Ablegung von Daten durchlaufen werden, soll im folgenden beschrieben werden.

Ein Anwenderprogramm ruft mittels DML das DBMS auf. Das DBMS ergänzt den Aufruf durch Informationen aus dem Schema und dem Unterschema und führt die notwendigen Ein- und Ausgabe-Operationen vom Betriebssystem her aus. Das Betriebssystem steuert die Ein- und Ausgabe und überträgt Daten zwischen peripherem und Pufferspeicher. Das DBMS überträgt die Daten vom Puffer in den Arbeitsbereich für das Anwenderprogramm. Alle notwendigen Datenumformungen, die durch Unterschiede zwischen

Schema und Unterschema bedingt sind, werden durch das DBMS dabei durchgeführt. Das DBMS liefert dem Anwender bzw. dem Anwenderprogramm Angaben über den Verlauf der Datenbearbeitung, und schließlich werden die Daten im Arbeitsbereich vom Anwenderprogramm in der Programmiersprache des Benutzers bearbeitet. Analog geht ein Ablegen in die Datenbasis vor sich.

Gewisse Zugriffsbeschränkungen müssen erlassen werden, wenn mehrere Anwender gleichzeitig von einer Datenbank Gebrauch machen können. Hierbei muß einerseits der Schutz der Daten gegen Nichtprivilegierte und andererseits der Schutz der Daten gegen Fehlermöglichkeiten gewährleistet sein. Das Problem der Sicherung der sogenannten Zugriffsrechte wird im DBTG-Schema durch sogenannte Sicherheitsschlösser und Sicherheitsschlüssel, die auf verschiedenen Datenebenen vergeben werden können, gelöst. Es soll in diesem Zusammenhang nicht näher darauf eingegangen werden.

Die Sicherung gegen Fehler, die dann auftreten können, wenn zahlreiche Anwender und mehrere Datenbasen nebeneinander und gleichzeitig arbeiten, soll durch die Einführung eines sogenannten Datenadministrators (Data-Administrator) geleistet werden. Er soll Funktionen der Verwaltung für die Mehrfachbenutzung übernehmen.

#### 4.1.3 Datenstrukturen

Im allgemeinen sind Sätze Gegenstand von Datenmanipulationen. Im DBTG-Bericht wird jedoch noch auf eine Stufe darunter zurückgegriffen, d.h. es wird Gebrauch von Satzstrukturen gemacht; diese gehen mit in das Schema

ein. Im folgenden seien kurz die DBTG-Begriffe aufgeführt.

Auf der untersten Ebene wird von Data-Item (Element) oder auch Atom gesprochen. Die nächste Stufe wird als Data-Group (Datenaggregat) bezeichnet. Man kann sie rekursiv wie folgt definieren: Ein Datenaggregat ist ein Verbund aus Elementen oder Verbunden. Es folgt der Vektor; er ist eine Wiederholungsgruppe aus Elementen. Die folgende Stufe ist die Repeating Group (wiederholter Verbund), die eine Wiederholungsgruppe aus Verbunden (Tabellen, Matrizen) darstellt. Es folgt dann der Record (Satz), der sich aus den bisher definierten Elementen zusammensetzen kann. Der Umgang mit Satzkomponenten fällt im wesentlichen in die Zuständigkeit von Anwenderprogrammen. Die Zusammenfassung von Sätzen fällt in die Zuständigkeit des DBMS.

Bei der Zusammenfassung von Sätzen werden im DBTG-Vorschlag die im folgenden aufgeführten Stufen unterschieden.

Auf der untersten Stufe steht die Area (Gebiet), die aus disjunktiven Mengen von Sätzen besteht. Es folgt die Data-Base (Datenbasis), eine Sammlung von Gebieten. Der Begriff Area ist sowohl logisch als auch physikalisch begründet, logisch in Hinsicht auf einen gemeinsamen Gebrauch der Sätze, physikalisch durch die Vorstellung von zusammenhängenden Bereichen im Hintergrundspeicher. Gebiete sowie Daten-Basen werden mit Bezeichnungen versehen. Als Set (Sammlung) wird eine Menge von Sätzen bezeichnet. Dies ist eine Menge von geordneten Paaren. Hierbei sind die folgenden Möglichkeiten gegeben:

- a) Jeder Satz kann beliebig vielen Sammlungen angehören, auch Sammlungen höherer Ordnung, wie z.B. hierarchischen Gebieten bzw. hierarchischen Strukturen. Da jeder Satz einem Gebiet angehören kann, kann es zu zahlreichen Mehrfachdarstellungen kommen.
- b) Ein Satz braucht keiner Sammlung anzugehören.
- c) Sätze einer Sammlung können in verschiedenen Gebieten liegen. Die Benennung, die sich eingebürgert hat, ist die des Owner-Satzes und des Member-Satzes. Sektoren oder Schlüssel in Sammlungen können immer nur Schlüssel aus Member-Sätzen sein (Sammlungsschlüssel), nie Schlüssel der Owner-Sätze. Sind die Verbindungen der Member-Sätze geordnet, so können die Bezeichnungen 'Next' bzw. 'Prior' als Schlüssel herangezogen werden. Sammlungsschlüssel müssen innerhalb einer Gesamtdatenbasis nicht eindeutig sein. In einem solchen Fall muß die Sammlung selbst identifiziert werden.

Schlüssel in Gebieten können nach folgenden Gesichtspunkten aufgebaut sein.

- a) Jedem Satz wird zum Zeitpunkt seiner Entstehung mit Hilfe einer DML-Anweisung ein eindeutiger Name, ein Daten-Basis-Schlüssel (Data-Base-Key), zugewiesen. Dieser bleibt während der Lebensdauer des Satzes erhalten, ist aber nur dem Anwender mittelbar über ein Attribut zugänglich. Man kann im strengen Sinne nicht von einem Bezeichner sprechen, der auf den Anwender ja zurückginge, sondern es handelt sich im Grunde um die Adresse eines entsprechenden Datenobjektes, die z.B. durch Umrech-

nungsverfahren auf eine physikalische Speicheradresse zurückgeführt wird.

- b) Man kann auch Rechenverfahren zur Abbildung des Schlüssels (Gebietsschlüssel) auf den Datenbasis-schlüssel verwenden.

Zur Auswahl einzelner Sätze werden im DBTG-Bericht Referenzvariablen vorgeschlagen. Diese werden als aktuelle Sätze (current records) bezeichnet. Folgende Typen sind möglich!

- a) unmittelbarer aktueller Satz (current of run unit), vergeben auf denjenigen Satz, auf den die letzte Operation angewendet wurde (vgl. Begriff 'Current Inverted File Record' in unserem System RS-II);
- b) aktueller Satz pro Satztyp (current of record), vergeben an denjenigen Satz, auf den innerhalb des gegebenen Satztyps die letzte Operation angewandt wurde. Da im allgemeinen mehrere Exemplare eines Satztyps existieren, erlaubt er im Zusammenhang mit dem Satznamen eine eindeutige Selektion;
- c) aktueller Satz pro Sammlungstyp (current of set). Es können mehrere Exemplare desselben vorgegebenen Sammlungstyps existieren. Bildet man die Menge aller Sätze dieser Sammlungsexemplare, so ist der aktuelle Satz dasjenige Element der Menge, auf das als letztes eine Operation angewandt wurde. Im Zusammenhang mit dem Sammlungsnamen (Sammlungstypname) wird hierdurch eine eindeutige Selektion des Sammlungsexemplars erreicht;

- d) aktueller Satz pro geöffnetem Gebiet (current of area), vergeben an denjenigen Satz, auf den innerhalb des gegebenen Gebiets die letzte Operation angewendet wurde.

#### 4.1.4 Schema und Unterschema

Ein Schema beschreibt die Datenbasis in einer durch den vorgegebenen Anwendungsbereich bestimmten Grundform. Die Anpassung an eine Anwendung erfolgt durch ein Unterschema, oder genauer, die Unterschiede zwischen dem Schema und dem Unterschema legen die Anwendungsform fest. Im folgenden werden die Abweichungen zwischen Schema und Unterschema aufgeführt.

Im Unterschema werden gegenüber dem Schema die Beschreibungen einzelner Strukturtypen wie Elemente, Aggregate, Sätze, Sammlungen, Gebiete weggelassen. Die Eigenschaften, d.h. Aufbau und Attribut derselben Aggregate und Sätze können im Schema und Unterschema voneinander abweichen. Gebiete, Satztypen und Sammlungstypen des Schemas können im Unterschema umbenannt werden. Schließlich können im Unterschema beliebige Stufen mit verschiedenen Zugriffsberechtigungen versehen werden. Im entsprechenden Anwendungsprogramm muß dafür die Legitimation (der Schlüssel) bereitgestellt werden. Zu jedem Schema kann es beliebig viele Unterschemata geben. Die Unterschemata sind völlig unabhängig voneinander, auch wenn sie mit ihren Beschreibungen keine der Datenbasen überdecken.

Beliebig viele Programme können jeweils von einem Unterschema Gebrauch machen. Diese Art der Festlegungen bieten Vorteile. Eine weitgehende Zugriffsbeschränkung

wird allein schon dadurch erreicht, daß ein Programm auf sein Unterschema beschränkt ist.

Die Anwenderprogramme sind weniger empfindlich gegen Umstrukturierung der Datenbasis oder Gerätewechsel, da sich diese Umstrukturierungen nur im Schema, nicht aber unbedingt im Unterschema auswirken müssen. Ferner ist es möglich, dieselbe Datenbasis sowohl in einer allgemein vereinbarten als auch in einer anwendungsspezifischen Terminologie zu beschreiben.

Definition und Übersetzung von Schemata und Unterschemata ist Aufgabe des Datenadministrators. Da bei den Unterschemata eine physikalische Umordnung im Arbeitsbereich notwendig sein kann, ist die Effizienz dadurch zu verbessern, daß das Schema im Hinblick auf einen im Mittel minimalen Umordnungs-Transformations-Aufwand entwickelt wird. Bei Hardware-Wechsel müssen zum Teil Transformationsprozeduren neu geschrieben bzw. ausgetauscht werden.

#### 4.1.5 Sammlungskonzept

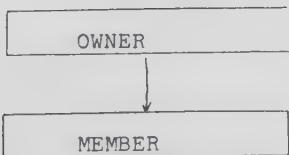
Wie wir in den bisherigen Ausführungen gesehen haben, besteht eine Schwierigkeit bei konventionellen Dateien darin, eine Zuordnung zwischen verschiedenen Satztypen zu definieren. Es existieren z.B. zwei verschiedene Satztypen, z.B. Autor und Buchtitel, und es taucht das Problem auf, daß während des Zugriffs in der Datei 1 auf ein Exemplar des records des Satztyps Autor noch auf ein Exemplar des Satztyps Buchtitel in Datei 2 zugegriffen werden muß. Zu diesem Zweck muß eine Beziehung definiert werden, die solche Zugriffe ermöglicht. Dies kann z.B. dadurch geschehen, daß in beiden Exemplaren beider Satztypen ein besonderes Kriterium, ein sogenannter Schlüssel, gespeichert wird, mit dessen

Hilfe nach dem Zugriff auf die Autorendatei auf die Buchtiteldatei zugegriffen werden kann, um dort Exemplare mit dem gleichen Schlüssel zu suchen.

Dieser Vorgang ist in den meisten Fällen allerdings nicht der optimale, da die Art der physikalischen Speicherung die Effektivität beeinflusst. Zudem muß der Benutzer jedesmal neben der logischen Beziehung auch die physikalischen Besonderheiten berücksichtigen und einbeziehen, was sehr bald zu der Forderung führt, ohne Kenntnis der physikalischen Speicherungsmethoden Beziehungen zwischen Records herstellen zu können, d.h. diese nur auf logischer Stufe zu definieren, und das Erkennen der Beziehungen, bzw. das Herstellen der Beziehungen dem System zu überlassen.

Es wird daher eine 1:n Relation eingeführt. Für unser Beispiel heißt das: einem Exemplar des Typs Autor werden mehrere Exemplare des Typs Buchtitel zugeordnet, wobei 'mehrere' hier jede natürliche Zahl und die Null bedeuten kann. Umgekehrt gibt es allerdings zu jedem Exemplar des Typs Buchtitel nur ein Exemplar des Typs Autor.

Man kann solche Relationen durch Schema-Zeichnungen darstellen, indem man Schemablöcke und gerichtete Verbindungen verwendet. Ihre Bezeichnungen im CODASYL-Vorschlag sind 'owner' und 'member'.



Diese Art der Darstellung nennt man Typ-Diagramm. Die Relation, die durch den Pfeil dargestellt wird, kann in der CODASYL-Terminologie als Sammlung bzw. set bezeichnet werden. Mit Hilfe von Schema-Diagrammen werden die Beziehungen zwischen Exemplaren von Satztypen dargestellt. In der obigen Darstellung könnte der Pfeil natürlich auch auf die owner zeigen. Dies wäre die zweite Art einer möglichen Beziehung. Ein Exemplar einer Sammlung besteht nur dann, wenn es ein Exemplar des owner-Typs gibt. Besteht ein Sammlungs-Exemplar nur aus einem owner, so ist die Sammlung leer.

Das Sammlungskonzept ist deshalb so wichtig, weil es als kleinster Baustein für alle in der Praxis gebräuchlichen Strukturen benutzt werden kann. Man kann damit sequentielle Strukturen, Baumstrukturen und Netze darstellen.

Sequentielle Strukturen können als geordnete Sammlungen von Record-Exemplaren dargestellt werden. Stehen Exemplare eines Record-Typs in keiner Relation zu Exemplaren des anderen Record-Typs, so läßt sich auf ihnen eine Struktur mit Hilfe eines Ordnungskriteriums (Schlüssel) definieren, ohne daß ein neuer Record-Typ eingeführt werden muß.

Eine Baumstruktur ist eine hierarchische Struktur, in der jedes Satz-Exemplar mit Ausnahme des ersten in Beziehung zu mehreren (einem oder keinem) Record-Exemplaren in der Hierarchie unter ihm und zu genau einem in der Hierarchie über ihm stehen kann. Ein einzelnes Sammlungs-Exemplar ist demnach schon ein (einstufiger) Baum. Der CODASYL-Vorschlag bietet drei Möglichkeiten, Baumstrukturen darzustellen:

- a) Eine Sammlung darf mehrere verschiedene Record-Typen als Member haben.
- b) Ein Record-Typ, der Member in einer Sammlung ist, darf gleichzeitig Owner einer anderen Sammlung sein.
- c) Ein Record-Typ darf Owner von mehreren Sammlungstypen sein.

Strukturen müssen nicht kreisfrei sein; so kann z.B. ein Record-Typ mit sich selbst in Beziehung stehen. Auf Exemplar-Ebene muß allerdings dann beachtet werden, daß ein Owner nicht zu seinen direkten oder indirekten Mitgliedern gehören darf.

Ein Netz erlaubt die Darstellung von  $m:n$  Relationen zwischen Record-Exemplaren. Dies wird ermöglicht, indem ein Record-Typ in mehreren Sammlungstypen als Member vorkommen kann. Es liegt hier der Fall einer polyhierarchischen Struktur vor, bei der ein Begriff mehr als einen Oberbegriff haben kann. Da jeder Sammlungstyp eine  $1:n$  Relation zwischen Owner und Member, aber nur eine  $1:1$  Relation zwischen Member und Owner ist, benötigt man in einer  $m:n$  Relation zwischen zwei Record-Typen zwei Sammlungstypen und drei Record-Typen.

Im folgenden sei die zusammenfassende Darstellung des Sammlungskonzeptes von SEIDEL (1975, S.13) wiedergegeben.

### Zusammenfassende Darstellung des Set-Konzepts

Das Set-Konzept erlaubt es, die Datenstruktur einer Datenbasis auf einer logischen Stufe zu definieren, unabhängig davon, wie die Daten dann physikalisch gespeichert werden. Der Benutzer kann alle Daten, die er benutzt, auf einer logischen Stufe ansprechen.

Zusammenfassend lassen sich folgende Eigenschaften von Sets (Settypen) und Setexemplaren angeben:

#### 1. Bezüglich Settypen:

Sets sind geordnete voreindeutige Relationen zwischen Record-Exemplaren.

In einem Schema zu einer Datenbasis kann eine beliebige Anzahl von Sets definiert werden.

Ein Record-Typ kann Owner von mehreren Sets sein.

Ein Record-Typ kann Member von mehreren Sets sein.

Ein Record-Typ kann Owner von mehreren Sets und Member von mehreren Sets sein.

#### 2. Bezüglich Setexemplaren:

Ein Record-Exemplar kann nicht Member in mehr als einem Exemplar desselben Sets sein.

Jedes Exemplar eines Sets muß ein Exemplar seines Ownerrecord-Typs haben, d.h. die Existenz eines Ownerrecord-Exemplars ist Voraussetzung für die Existenz eines Setexemplars.

Existiert nur ein Ownerrecord-Exemplar, aber kein Memberrecord-Exemplar, so heißt das Setexemplar leer.

Jedes Setexemplar kann zu einem bestehenden Ownerrecord-Exemplar mehrere Exemplare von jedem im Schema definierten Memberrecord-Typ enthalten.

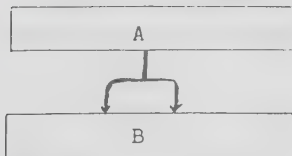
Jedes Element der Membermenge hat genau ein Owner-record-Exemplar in diesem Set.

Die Einordnung von Memberrecord-Exemplaren in ein Setexemplar geschieht nach im Schema spezifizierten Ordnungskriterien.

#### 4.1.6 Erweiterung des Sammlungskonzepts und Schwierigkeiten bei der Anwendung auf komplexe Datenstrukturen

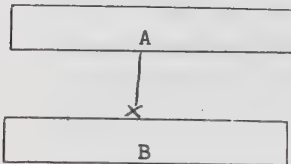
Wie vorhin aufgezeigt, können mit Hilfe der einfachen Datenbeziehung auch komplexe Datenstrukturen dargestellt werden. In einigen Fällen ist es dabei allerdings notwendig, zusätzliche Record- und Sammlungstypen einzuführen, so z.B. bei der Darstellung einer m:n Relation.

Häufig wird gefordert, daß in einem Datenbanksystem eine m:n Relation zwischen Record-Exemplaren erlaubt wird. Dies ist gleichbedeutend mit der Forderung nach invertierten Listen.

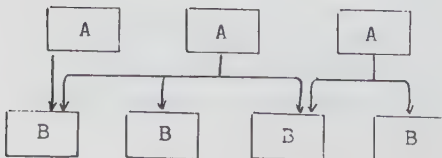


Dargestellt wird ein Beispiel für eine 2:n Relation. Dies bedeutet, daß jedes Exemplar des Typs B höchstens zu sovielen Member-Mengen (mit je einem Owner) gehören kann, wie Pfeile zu ihm hinführen, in diesem Fall also zwei.

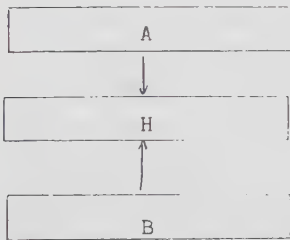
Allgemein gilt, daß es zu jedem Exemplar des Record-Typs B mehrere Owner-Exemplare geben kann und zu jedem Owner-Exemplar unbestimmt viele Member-Exemplare. Solche Relationen werden als x-Relationen gekennzeichnet.



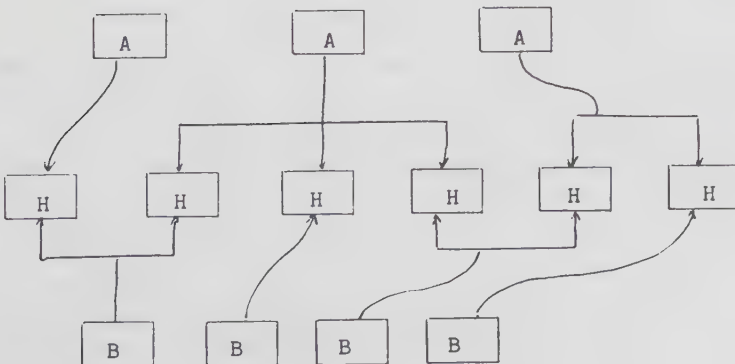
Möchte man z.B. darstellen, daß eine Diagnose bei vielen Patienten vorkommt, und auch, daß ein Patient beliebig viele Diagnosen haben kann, so ist dies nach dem bisher Dargelegten nicht unmittelbar möglich, da es die Bedingung verletzt, daß ein Member nur einen Owner haben darf. Demnach müßte eine solche x-Relation wie folgt dargestellt werden:



Diese Struktur lässt sich durch Einführung eines Hilfsblocks oder Records auf eine Grundstruktur zurückführen. Der Record-Typ H wird in diesem Fall nur zur Strukturdarstellung gebraucht; er enthält selbst keinerlei Daten. Trotzdem muß er z.B. in der Datendefinitionssprache (DDL) definiert werden, damit das DBS die Struktur erkennen kann.

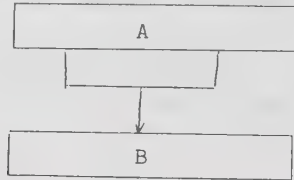


Zum besseren Verständnis dieser Struktur sei noch einmal das Exemplar der x-Relation unter Einführung der Hilfsblöcke dargestellt.

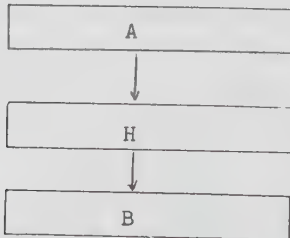


Pro Exemplar des Typs A benötigt man sovieler Hilfs-exemplare H wie Exemplare des Typs B zu A in Relation stehen.

Ebenfalls nicht unmittelbar darstellbar ist die folgende Struktur:

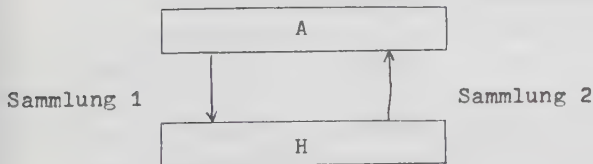


Es betrifft den Fall, daß es zu jedem Exemplar des Typs A unbestimmt viele disjunkte Member-Mengen gibt, zu denen es Owner ist. Diese Beziehung wird auch als v-Relation gekennzeichnet. Auch diese Struktur läßt sich erklären als Abkürzung für die folgende, die ebenfalls einen Hilfsblock H verwendet. Dabei sind beide Strukturen allerdings nicht streng äquivalent.

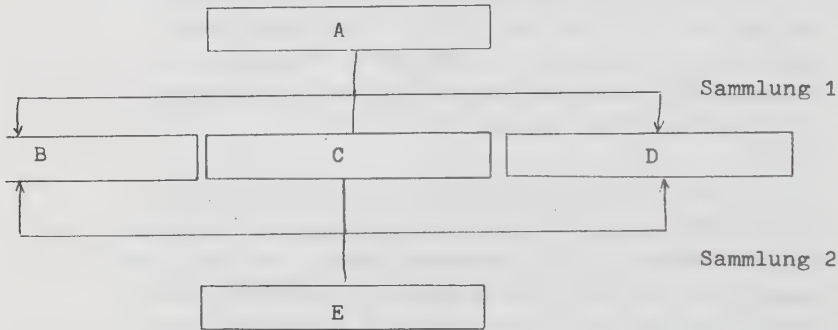


Hier müssen pro Exemplar A so viele Hilfsexemplare H eingeführt werden, wie es Member-Mengen zu A gibt. Bei der Implementierung einer solchen Struktur ist allerdings ein erheblicher Aufwand an Speicherplatz zur Manipulation notwendig. Es sollte daher zur Regel werden, daß komplexe Strukturen innerhalb von Exemplaren die Ausnahme sind. Durch Verletzung des Redundanzprinzips kann eine leichtere Manipulation der Daten erreicht werden.

Eine Struktur, die in CODASYL ebenfalls nur schwer zu verwirklichen ist, ist die elementarste Form einer Baumstruktur, d.h.: ein Record-Typ ist Owner und Member des gleichen Sammlungstyps. Eine Möglichkeit der Darstellung zeigt die folgende Zeichnung. Auch hier wird ein Hilfsrecord-Typ H benutzt, wobei hier jedoch die Anzahl von Exemplaren des Typs H auf eins beschränkt ist. Durch Sammlung 1 wird eine 1:1 Relation dargestellt. Eine Umwandlung in diese Struktur bietet allerdings nur den Vorteil, daß man auch hier ein Exemplar des Typs A zu sich selbst in Relation setzen kann, enthält aber gleichzeitig den Nachteil, daß man einen Sammlungstyp und einen Recordtyp mehr benötigt.



Eine weitere Einschränkung der erlaubten Strukturen besteht darin, daß ein Sammlungstyp nur einen Record-typ als Owner haben darf.



Die obige Struktur stößt bei der Realisierung ebenfalls auf Schwierigkeiten, da der Record-Typ E in der Sammlung 2 drei Record-Typen als Owner haben kann, wobei ein Sammlungs-Exemplar jedoch nach wie vor nur einen Owner hat. Diese Struktur läßt sich allerdings in eine Grundstruktur überführen, indem man an Stelle einer Sammlung (Sammlung 2) drei verschiedene Sammlungen definiert. Zeigt sich in der Praxis, daß solche Strukturen sehr häufig benutzt werden, so ist man u.E. gehalten, diese Strukturen als Grundstrukturen in das CODASYL-Konzept einzubetten.

#### 4.2 Relationalmodell

Das Relationalmodell mit dem mathematischen Begriff der Relation als Grundlage wurde im wesentlichen von CODD (1970) entwickelt. Mit Implementierungen des Re-

lationalmodells begann man erst in jüngster Zeit (ASTRAHAN et al. System R, 1976). Zwischen den Anhängern des CODASYL-DBTG-Vorschlags und des Relationalmodells bestehen erhebliche Kontroversen im Hinblick auf Fragen der praktischen Anwendbarkeit (z.B. Schnelligkeit der Ausführung) und der logischen Stringenz. Eine Synthese zwischen beiden Positionen zeichnet sich jedoch innerhalb der sogenannten Schichten-Architektur ab.

#### 4.2.1 Begriffe

Das Relationalmodell kommt im wesentlichen mit zwei Begriffen aus (im Gegensatz zum DBTG-Modell, das ja eine Fülle von Begriffen verlangt), und zwar sind dies erstens der Begriff der Relation und zweitens der Begriff des Schlüssels.

Der Schlüssel wird ausschließlich als identifizierender Schlüssel verstanden und nicht in der Funktion, beliebige Attribute in knapper Form wiederzugeben, wie dies bei klassifizierenden Schlüsseln der Fall ist. Relationen in Klammerschreibweise, z.B.  $R(x_1, x_2, x_3, \dots, x_n)$ , werden als n-stellige Relationen eingeführt.

Die Mengennamen sind Attributnamen; die Mengen bestehen aus Attributwerten.

Im folgenden sei eine Relation PATIENT dargestellt; die Attributnamen (domain) der Relation PATIENT sind PATIENTENNUMMER, PATIENTENNAME, PATIENTENVORNAME und KRANKENKASSE. Wir können schreiben PATIENT (PATIENTENNUMMER, PATIENTENNAME, PATIENTENVORNAME, KRANKENKASSE). Attributwerte einer Relation werden für gewöhnlich in einer Tabelle eingeordnet. Ein n-Tupel ist dann eine Zeile, ein Tupel entspricht einem Satz, und ein Attri-

but entspricht einem Feld (Wort), die gesamte Relation entspricht einer Datei.

Eine wichtige Eigenschaft einer Relation ist, daß keine Zeile mehrfach vorkommt. Wäre dies der Fall, so läge eine unnötige Redundanz vor. Wenn aber alle Tupel (Zeilen) unterschiedlich sind, dann gibt es mindestens ein Attribut, das ein Tupel eindeutig identifiziert. Dieses Attribut oder die Attributgruppe ist der Schlüssel. Von einem Schlüssel fordert man, daß er eine Minimum-Eigenschaft hat. Nur Attribute, die wirklich identifizieren, sind Schlüssel. So ist z.B. PATIENTENNUMMER und PATIENTENNAME zusammen genommen kein Schlüssel, PATIENTENNUMMER für sich allein genommen ist ein Schlüssel.

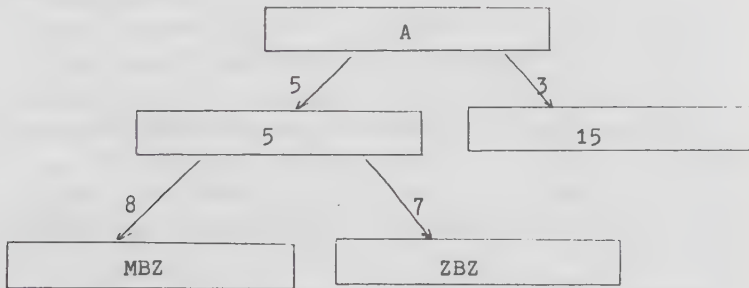
Mehrere Attribute zusammen haben nur dann Schlüsseligenschaft, wenn die einzelnen Teile des Schlüssels nur einmal vorkommen, d.h. also identifizierende Eigenschaften haben.

Attribute, die eine identifizierende Eigenschaft haben, werden Schlüsselkandidaten genannt. Das Attribut, dem die Identifizierungsaufgabe tatsächlich zugeschrieben wird, nennt man Primärschlüssel. Es ist im allgemeinen das Attribut mit der kürzesten Feldlänge. Dies ist häufig eine laufende Nummer, Personalnummer, usw.

Im Relationalmodell müssen alle Werte eines Schlüsselattributs (Primärschlüssel) definiert sein. Für die Darstellung und die Kommunikation über Relationen ist es unerheblich, in welcher Ordnung die Spalten aufgeführt werden, es muß nur in einer Relation der Attributname eindeutig sein. Der Relationsname muß über das ganze Datenbanksystem eindeutig sein.

Es ist für den Begriff der Relation nicht notwendig, daß die Tupel nach irgendeinem Attribut (Ordnungskriterium) geordnet sind. Um zu zeigen, daß der Begriff Tupel oder Entity ein ganz allgemeiner Begriff ist und ähnlich abstrakt wie der Begriff Element einer Menge, sei z.B. folgende Relation dargestellt, die den Namen 'Struktur' trägt.

Eine Abteilung bestehe aus 5 Stationen und 3 Intensivstationen. Jede dieser 5 Stationen habe 8 Mehrbettzimmer und 7 Zweibettzimmer.



Die Struktur können wir wie folgt schreiben:  
Struktur (übergeordneter Teil, untergeordneter Teil, Anzahl).

Die Attribute, übergeordneter Teil und untergeordneter Teil, bilden zusammen den Schlüssel.

#### 4.2.2 Normalisierung

Das CODD'sche Relationenmodell geht davon aus, daß für alle Arten von Benutzern die einfache Tabelle (Relation) eine sehr naheliegende Anordnung von Daten ist. Es läßt sich zeigen, daß Baumstrukturen und Netzwerke in einfacher Weise ohne Verlust von Information mit Hilfe von einfachen Relationen dargestellt werden können. Im folgenden sei der Prozeß der Normalisierung, d.h. der Prozeß der Überführung von Abhängigkeitsformen (Hierarchien, Netzwerke) in einfache Relationen dargestellt (WEDEKIND, 1975).

Als normalisierte Relation wird eine Relation bezeichnet, die aus einfachen Attributmengen besteht. Eine einfache Attributmenge ist hierbei eine Menge von Werten, die nicht weiter zerlegbar ist. Relationen, die keine einfachen Attributmengen enthalten, werden unnormalisierte Relationen genannt. Die Überführung einer unnormalisierten Relation in eine oder mehrere normalisierte Relationen ohne den Verlust von Information wird Normalisierung genannt. 'Ohne Verlust von Information' bedeutet in diesem Zusammenhang, daß die Normalisierung wieder rückgängig gemacht werden kann.

Das Prinzip der Normalisierung soll an folgendem Beispiel erläutert werden.

| UNNORM  |            |        | NORM    |        |        |
|---------|------------|--------|---------|--------|--------|
| ARZT-NR | BEHANDLUNG |        | ARZT-NR | GOÄ-NR | ANZAHL |
|         | GOÄ-NR     | ANZAHL |         |        |        |
| 1       | 1          | 3      | 1       | 1      | 3      |
|         | 2          | 4      | 1       | 2      | 4      |
|         | 3          | 2      | 1       | 3      | 2      |
| 2       | 1          | 4      | 2       | 1      | 4      |
|         | 2          | 5      | 2       | 2      | 5      |
| 3       | 3          | 1      | 3       | 3      | 1      |
|         | 4          | 3      | 3       | 4      | 3      |
|         | 5          | 8      | 3       | 5      | 8      |

Die Arzt-Nummer und die GOÄ-Nummer (Nummer in der Gebührenordnung für Ärzte) stehen hier für eine größere Anzahl von Attributen. Die Relation UN-NORM (ARZTNUMMER, BEHANDLUNG (GOÄ-NR, ANZAHL)) ist unnormalisiert. Die Attributmenge BEHANDLUNG ist nicht einfach, sondern setzt sich aus den Attributmengen GOÄ-NR und ANZAHL zusammen. Die Relation NORM ist eine äquivalente, normalisierte Relation. Durch die Wiederholung der Attributwerte für die Arztnummer wird in der normalisierten Form die hierarchische Zuordnung Arzt - Behandlung aufrechterhalten, die nichteinfache Attributmenge BEHANDLUNG verschwindet, indem sie in die Bestandteile GOÄ-NR und ANZAHL aufgelöst wird.

Im folgenden wird die Normalisierung in ihren einzelnen Schritten dargestellt. Ausgangspunkt ist die folgende Relation:

ARZT (ARZTNUMMER, NAME)  
THERAPIE (ARZTNUMMER, GOÄ-NR, ANZAHL)

Das Beispiel soll noch erweitert werden, indem der Relation THERAPIE noch ein Datum hinzugefügt wird:

THERAPIE (DATUM, BEHANDLUNG (GOÄ - NUMMER, ANZAHL))

Die Relation THERAPIE ist jetzt eine zusammengesetzte (compound) Gruppe. Da während eines Tages mehrere Behandlungen durchgeführt werden können, ist die Gruppe BEHANDLUNG auch wiederholend. Damit existieren also zwei Wiederholungsgruppen auf verschiedenen Ebenen. Einmal ist THERAPIE eine Wiederholungsgruppe, zum anderen ist BEHANDLUNG eine Wiederholungsgruppe im Ver-

hältnis zu DATUM. Geschrieben wird dies wie folgt:

ARZT (ARZTNUMMER, NAME)

THERAPIE (DATUM, BEHANDLUNG (GOÄ-NR, ANZAHL))

In Form einer Tabelle kann dies wie folgt aussehen:

| ARZTNUMMER | NAME   | DATUM   | BEHANDLUNG |        |
|------------|--------|---------|------------|--------|
|            |        |         | GOÄ-NR     | ANZAHL |
| 1          | MÜLLER | 13.9.76 | 1          | 3      |
|            |        |         | 5          | 4      |
|            |        | 14.9.76 | 2          | 3      |
|            |        |         | 1          | 1      |
| 2          | MEIER  | 13.9.76 | 3          | 14     |
|            |        |         | 6          | 3      |
|            |        | 15.9.76 | 3          | 12     |

Der Normalisierungsprozeß sieht wie folgt aus:

1. Es wird ein Schema in einer Form angegeben, in der keine doppelten Klammern vorkommen, z.B.:

ARZT (ARZTNUMMER, NAME)

THERAPIE (DATUM, BEHANDLUNG)

BEHANDLUNG (GOÄ-NR, ANZAHL)

2. In dieser Relationen-Schreibweise werden die Primärschlüssel durch Unterstreichungen gekennzeichnet. Es sind dies Arztnummer, Datum und GOÄ-Nummer (siehe 1.).
3. Man beginnt mit der Relation, die an der Baumwurzel steht. Der Primärschlüssel dieser Relation wird in die untergeordneten Relationen eingefügt. Der Primärschlüssel der so erweiterten Relation

besteht aus dem Primärschlüssel vor der Erweiterung und dem Primärschlüssel, der aus der übergeordneten Relation stammt. In der übergeordneten Relation können nun alle nichteinfachen Attribute gestrichen werden. Die Wurzel des Baumes ist normalisiert und kann als solche entfernt werden. Es bleibt bestehen:

THERAPIE (ARZTNUMMER, DATUM, BEHANDLUNG)  
BEHANDLUNG (GOÄ-NR, ANZAHL)

4. Der eben beschriebene Schritt wird für alle Unterbäume durchgeführt. Das Ergebnis dieser Normalisierung ist die folgende Relation:

BEHANDLUNG (ARZTNUMMER, DATUM, GOÄ-NR, ANZAHL)

Wesentliches Kennzeichen des Relationenmodells ist es, daß Wiederholungsgruppen fehlen. Dieser Normalisierungsprozeß wurde zum erstenmal von CODD (1970) beschrieben. Ein großer Teil des Relationenmodells beschäftigt sich mit der Normalformenlehre. Sie ist in der Denkweise der bisherigen Datenverarbeitung eine Theorie des Satzaufbaues unter Verwendung der Begriffe Sätze (records) und Dateien (files). Man kennt auch eine zweite und eine dritte Normalform, auf die hier aber nicht näher eingegangen werden soll. Es sei lediglich gesagt, daß in der zweiten bzw. dritten Normalform Attribute, d.h. Felder wieder zu Sätzen zusammengefaßt werden, die auch von der Anwendung her zusammengehören.

Es ist auch möglich, eine Normalisierung durch Synthese zu erreichen, wie es WANG und WEDEKIND (1974) gezeigt haben. Die CODD'schen Normalformen werden durch

einen Zerlegungsprozeß gewonnen, während WANG und WE-DEKIND synthetisch vorgehen. Sie lösen das Segmentierungsproblem unter dem Aspekt des Entwurfes.

#### 4.2.3 Operationen mit Tupeln

Die möglichen Darstellungen von Tupeln sollen an den folgenden Beispielen noch einmal aufgezeigt werden. Gegeben ist die Relation BELEGUNG. Beide Darstellungen sind identisch.

| BELEGUNG |         | BELEGUNG |         |
|----------|---------|----------|---------|
| PATIENT  | STATION | PATIENT  | STATION |
| 1        | 1       | 1        | 1       |
| 2        | 1       | 2        | 3       |
| 3        | 2       | 1        | 2       |

Eine ebenfalls wählbare Darstellungsform ist die der Ereignistabelle:

|         |   | PATIENT |      |      |
|---------|---|---------|------|------|
|         |   | 2       | 3    | 1    |
| STATION | 2 | nein    | ja   | nein |
|         | 1 | ja      | nein | ja   |

Für jedes Tupel einer Relation kann ein neues Tupel gebildet werden, das nur einige Elemente enthält (Projektion).

| BELEGUNG |         |         |
|----------|---------|---------|
| STATION  | PATIENT | STATION |
| 1        | 1       | 1       |
| 2        | 3       | 2       |
| 1        | 2       |         |

Aus der Menge der Tupel kann nun auch eine Untermenge ausgewählt werden, die wiederum eine Relation bildet. Z.B. kann man aus der Relation BELEGUNG alle Tupel auswählen, bei denen die Bedingung gilt: STATION 1.

BELEGUNG

| STATION | PATIENT | STATION | PATIENT |
|---------|---------|---------|---------|
| 1       | 1       | 1       | 1       |
| 2       | 3       | 1       | 2       |
| 1       | 2       |         |         |

In beiden Beispielen kann das Ergebnis mit Hilfe eines speziellen Operators, einem 'Join'-Operator, erreicht werden. Er kann ebenfalls dazu benutzt werden, Informationen, die über mehrere Relationen verstreut sind, in eine zusammenzuführen.

Als weiteres Beispiel sei die Relation PROBLEM gegeben.

PROBLEM

| PATIENT | BEHANDLUNG |
|---------|------------|
| 1       | 4          |
| 2       | 8          |
| 3       | 10         |

Beide Relationen können über das Attribut PATIENT verbunden werden:

| PATIENT | BEHANDLUNG | PATIENT | STATION |
|---------|------------|---------|---------|
| 1       | 4          | 1       | 1       |
| 2       | 8          | 2       | 2       |
| 3       | 10         | 3       | 1       |

Eine Verbindung (join) zweier Relationen kann allgemein in zwei Stufen hergestellt werden..

1. "full quadratic join" (TODD, 1976), Cartesisches Produkt jedes Tupel der einen Relation wird mit jedem Tupel der anderen Relation verkettet. Die Anzahl der Tupel der Ergebnisrelation ist das Produkt der Anzahlen in den beiden zu verbindenden Relationen.

| PATIENT | BEHANDLUNG | PATIENT | STATION |
|---------|------------|---------|---------|
| 1       | 4          | 1       | 1       |
| 1       | 4          | 2       | 1       |
| 1       | 4          | 3       | 1       |
| 2       | 8          | 1       | 1       |
| 2       | 8          | 2       | 2       |
| 2       | 8          | 3       | 1       |
| 3       | 10         | 1       | 1       |
| 3       | 10         | 2       | 2       |
| 3       | 10         | 3       | 1       |

2. Anschließende Auswahl

Es werden dabei die Tupel ausgewählt, bei denen die Werte des Verbindungsfeldes (PATIENT) identisch sind.

| PATIENT | BEHANDLUNG | PATIENT | STATION |
|---------|------------|---------|---------|
| 1       | 4          | 1       | 1       |
| 2       | 8          | 2       | 2       |
| 3       | 10         | 3       | 1       |

Die Vorschrift, eine Verbindung herzustellen, kann man auch wie folgt formulieren:

Verbinde jedes Tupel der ersten Relation mit dem Tupel in der zweiten Relation, das im Verbindungsfeld den gleichen Wert aufweist. Der Umfang einer Verbindung ist schwer abzuschätzen (zwischen 0 und dem Produkt beider Relationen).

Üblicherweise liegt der Umfang in der Größenordnung der größeren der beiden zu verbindenden Relationen.

Das eigentliche Problem ist, bei der Implementierung schnell arbeitende Verfahren zu finden.

Bevor auf die Implementierung von Verbindungen eingegangen wird, seien zunächst die Möglichkeiten genannt, wie Relationen selbst gespeichert werden:

- a) in einem sequentiellen File (ein Tupel in einem Record)
- b) in sequentiellen Files, wobei eine Spalte (Domäne) in einem (oder mehreren) Record(s) abgesetzt wird
- c) in invertierten Dateien (s.o.)
- d) in Bitmatrizen (s.o.).

#### 4.2.4 Untersuchung der Implementierungsmöglichkeiten des "join"-Operators in einer gebräuchlichen Programmiersprache (F IV)

Es werden im folgenden die Möglichkeiten besprochen, wie eine Verbindung (join) realisiert werden kann, und zwar an folgendem Beispiel:

| Relation I | Relation II |
|------------|-------------|
| A X        | L Y         |
| A Y        | M X         |
| B X        | M Y         |

Die Relationen sollen über die zweite Spalte verbunden werden.

a) Bilden des Cartesischen Produktes

In der einfachsten Form der Bildung eines Cartesischen Produkts wird ein File sequentiell gelesen, und so werden Paar für Paar verglichen. Vorteil dieser Methode ist bei sortierten Ausgangs-Files, daß ebenfalls ein sortierter Ergebnis-File entsteht.

|   |   |  |   |   |                |
|---|---|--|---|---|----------------|
| A | X |  | L | Y | Ausgangs-Files |
|---|---|--|---|---|----------------|

|   |   |  |   |   |  |
|---|---|--|---|---|--|
| A | Y |  | M | X |  |
|---|---|--|---|---|--|

|   |   |  |   |   |  |
|---|---|--|---|---|--|
| B | X |  | M | Y |  |
|---|---|--|---|---|--|

Bilden des Cartesischen  
Produktes

|   |   |   |   |
|---|---|---|---|
| A | X | L | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | X | M | X |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | X | M | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | Y | L | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | Y | M | X |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | Y | M | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| B | X | L | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| B | X | M | X |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| B | X | M | Y |
|---|---|---|---|

Auswahl

|   |   |   |   |
|---|---|---|---|
| A | X | M | X |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | Y | L | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | Y | M | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| B | X | M | X |
|---|---|---|---|

b) Zusammenführen\_von\_sortierten\_Files

Ein File wird sequentiell Record für Record gelesen, für jeden gelesenen Record braucht vom anderen File nur jeweils ein kleiner Abschnitt gelesen zu werden. Wenn im ersten File Werte des Verbindungsfeldes nur jeweils einmal vorkommen (Primärschlüssel), braucht der zweite File nur einmal sequentiell gelesen zu werden.

Ist das Verbindungsfeld des ersten Files nicht der Primärschlüssel, dann müssen einige Teile des zweiten Files zweimal gelesen werden. Man markiert auf eine beliebige Weise die Stelle des zweiten Files, an der man sich befindet, und kann nach einigen Verarbeitungsdurchgängen wieder auf diese Marke zurückgehen.

Sehr oft wird man sortierte Files vorfinden; ist dies nicht der Fall, kann (muß) vorher sortiert werden.

Sind die Records nicht zu lang, stellt die Zusammenführung die schnellste Möglichkeit dar, eine Verbindung zu schaffen.

|   |   |   |   |                |
|---|---|---|---|----------------|
| A | X | L | Y | Ausgangs-Files |
|---|---|---|---|----------------|

|   |   |   |   |
|---|---|---|---|
| A | Y | M | X |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| B | X | M | Y |
|---|---|---|---|

Sortieren

|   |   |   |   |
|---|---|---|---|
| A | X | M | X |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| B | X | L | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | Y | M | Y |
|---|---|---|---|

Zusammenführen

|   |   |   |   |
|---|---|---|---|
| A | Y | M | X |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| B | X | M | X |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | Y | L | Y |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| A | Y | M | Y |
|---|---|---|---|

c) Verbinden von Indizes

Betrachtet werden Files mit zwei Feldern pro Record. Ein Feld enthält einen Wert eines bestimmten Feldes des Files, der invertiert wurde. Das andere Feld enthält einen Record-Identifizier des betreffenden Records.

Zur Verfügung stehen die oben beschriebenen invertierten Files. Sie seien nach den Werten sortiert. Beide werden über die Feldwerte verbunden (Verbindungsfeld), und die Feldwerte werden weggelassen. Das Ergebnis ist ein erster Zwischen-File mit Records von je zwei Feldern: Im ersten befindet sich der Record-Identifizier vom ersten File, im zweiten der vom zweiten File. Anschließend wird nach dem ersten oder zweiten Feld sortiert (d.h. die Record-Identifizier werden sortiert).

Dieser Zwischenfile wird mit einem Ausgangsfile recordweise zusammengefügt. Angenommen, der erste Zwischenfile wird nach dem zweiten Record-Identifizier sortiert und mit dem zweiten Ausgangsfile zusammengefügt, indem der Record-Identifizier durch den betreffenden Record ersetzt wird, ist das Ergebnis ein zweiter Zwischenfile, der den Record-Identifizier des ersten Ausgangsfiles und Records des zweiten Ausgangsfiles enthält. (In praxi werden sich die Sortierungen auf Adressen von Records beziehen.)

Dieser zweite Zwischenfile wird dann nach dem ersten Record-Identifizier sortiert. Danach wird wie oben beschrieben der erste Ausgangsfile mit diesem zweiten Zwischenfile zusammengefügt. Es entsteht der Ergebnisfile.

In manchen Fällen wird eine Sortierung nach einem bestimmten Feld schneller sein als eine teilweise Invertierung. Die eben beschriebene Methode hat dann große Vorteile, wenn invertierte Listen bereits existieren, aber die Ausgangsfiles nicht in geeigneter Sortierreihenfolge vorliegen.

Bei sehr großen Ausgangsfiles kann eine teilweise Invertierung schneller zum Ziel führen als eine Sortierung nach einem bestimmten Feld, da das Zusammenfügen von invertierten Listen sehr schnell ist.

|   |   |   |   |               |
|---|---|---|---|---------------|
| A | X | L | Y | Ausgangsfiles |
| A | Y | M | X |               |
| B | X | M | Y |               |

|   |   |   |   |                                    |
|---|---|---|---|------------------------------------|
| X | 1 | X | 2 | Invertierung nach dem zweiten Feld |
| X | 3 | Y | 1 |                                    |
| Y | 2 | Y | 3 |                                    |

|   |   |   |   |               |
|---|---|---|---|---------------|
| X | 1 | X | 2 | Zusammenfügen |
| X | 3 | X | 2 |               |
| Y | 2 | Y | 3 |               |
| Y | 2 | Y | 3 |               |

|   |   |                          |
|---|---|--------------------------|
| 1 | 2 | Fortlassen der Feldwerte |
| 3 | 2 |                          |
| 2 | 1 |                          |
| 2 | 3 |                          |

|   |   |                                                |
|---|---|------------------------------------------------|
| 2 | 1 | Sortieren nach dem zweiten Record-Identifizier |
| 1 | 2 |                                                |
| 3 | 2 |                                                |
| 2 | 3 |                                                |

|   |   |   |                                            |
|---|---|---|--------------------------------------------|
| 2 | L | Y | Zusammenfügen mit dem zweiten Ausgangsfile |
| 1 | M | X | 1. L Y                                     |
| 3 | M | X | 2. M X                                     |
| 2 | M | Y | 3. M Y                                     |

|   |   |   |                                               |
|---|---|---|-----------------------------------------------|
| 1 | M | X | Sortieren nach dem ersten Record-Identifizier |
| 2 | L | Y |                                               |
| 2 | M | Y |                                               |
| 3 | M | X |                                               |

|   |   |   |                                           |
|---|---|---|-------------------------------------------|
| 1 | M | X | Zusammenfügen mit dem ersten Ausgangsfile |
| 2 | L | Y | 1. A X                                    |
| 2 | M | Y | 2. A Y                                    |
| 3 | M | X | 3. B X                                    |

|   |   |   |   |              |
|---|---|---|---|--------------|
| A | X | M | X | Ergebnisfile |
| A | Y | L | Y |              |
| A | Y | M | Y |              |
| B | X | M | X |              |

An diesem Beispiel ist die leichte Implementierbarkeit (z.B. in FORTRAN) sofort ersichtlich. Ob die in dieser Weise zur Verfügung gestellten Operatoren auch effektiv arbeiten, ist eine noch weiter zu untersuchende Frage.

## II. Praktische Anwendungen von Datenbank-Organisations- prinzipien

### 1 Thesen

Da man davon ausgehen muß, daß umfassende und komplexe Datenbanksysteme auf kleineren Rechnern nicht implementiert werden können, soll im folgenden versucht werden, ein alternatives Vorgehen zu beschreiben.

Dabei spielen Gedanken über Modularität, (Baukastenprinzip), und Portabilität eine besondere Rolle.

#### These 1

So wie bei Hardwaresystemen (Mikroprozessor) ein System durch Hinzufügen von weiteren Einheiten (Platinen) erweitert werden kann, muß eine Datenverwaltung aus portablen Systemelementen schrittweise zusammengesetzt werden können (Prinzip: "soviel wie nötig", TOOL-POOL).

Sehr viele Datenbanksysteme erweisen sich für eine spezielle Anwendung bzw. einen spezifischen Anwendungsbe-  
reich als überdimensioniert, da sie Möglichkeiten bieten, die zwar bezahlt werden müssen, nicht aber genutzt werden. So zeigt sich z.B., daß Hierarchien der Stufe 3 völlig den Anforderungen im medizinischen Bereich genügen und daß an den Komfort einer Abfragesprache nur geringe Anforderungen gestellt werden (LIOU, 1977).

Das zentrale zu lösende Problem in diesem Zusammenhang ist die Zerlegung von Funktionen in autonome Teilfunktionen.

## These 2

Nicht alle Systemkomponenten eines geplanten (an spezifische Bedürfnisse angepaßten) Systems müssen sofort zur Verfügung stehen; vielmehr ist ein schrittweiser Aufbau im Rahmen eines Gesamtkonzepts zu fordern.

Sowohl von der Systemerstellung und von der Systemte-  
stung, als auch von der Einführung und Gewöhnung der  
Benutzer an ein System her ist eine Aufteilung des Sy-  
stems in Grundfunktionen, (Standard-) Erweiterungen  
und Zusätze zu fordern (HENSELMANN, 1977).

Bei Realisierungen von Datenbanksystemen ist die Kom-  
munikation der Teilkomponenten über eine Instanz  
(Schnittstelle) zu führen, die ihrerseits nicht starr  
definiert ist, sondern selbst wieder parametrisiert  
sein sollte. Ein System kann bereits getestet werden,  
wenn für einige Bausteine nur sogenannte Dummies vor-  
handen sind (HIPO - IBM).

Hier entsteht als zentrales Problem das der Schnitt-  
stellendefinition.

## These 3

Bausteine sollten grundsätzlich in einer strukturier-  
ten allgemeinen Sprache erstellt werden. Ihre Übertra-  
gung in eine problem- oder maschinenorientierte Spra-  
che ist erst in der Implementierungsphase notwendig.

Schwierigkeiten ergeben sich bei der Auswahl der zuge-  
lassenen Sprachelemente (Rekursion, Bit-Byte-Opera-  
tionen, Verwendung von hierarchischen Datenstrukturen  
(PL/1-Typ, COBOL-Typ)).

Wir meinen nicht, daß es möglich ist, allgemeine standardisierte gültige Module z.B. in FORTRAN zu erstellen, was nicht bedeutet, daß auf eine Realisierung in Standard-FORTRAN verzichtet werden muß (Im Anhang dieser Arbeit befinden sich verschiedene in FIV realisierte Algorithmen.).

Das zentrale Problem hier ist die Auswahl von Sprach-elementen einer allgemeinen strukturierten Sprache.

#### These 4

Der Aufbau einer Datenbank geschieht nach einer Checkliste durch ein spezifisch zusammengesetztes Team, in dem die Benutzerseite stark vertreten ist.

Ausgehend von einer umfassenden Checkliste über Datenbankkriterien für alle denkbaren Anwendungsfälle werden durch das Team die relevanten Punkte für den zu lösenden Fall ausgewählt und ein integrierendes Konzept entwickelt (logische, zeitliche Abhängigkeiten). Die Bausteine aus dem Tool-Pool, die diese Kriterien bzw. Anforderungen abdecken, werden ebenfalls zusammengestellt. Ein "Koppelungs-Plan" gibt dann Auskunft über das Zusammenfügen der Bausteine zu dem konzipierten System.

Als Probleme, die hier im Mittelpunkt stehen, sind die Vollständigkeit der Checkliste und die Zusammensetzung des Teams zu nennen.

2 Überlegungen zur Auswahl einer Beschreibungssprache  
unter Berücksichtigung ihrer Übertragbarkeit in  
FORTRAN

Als grundsätzliche Möglichkeiten bieten sich an:

- a) Formulierung in einer allgemeinen strukturierten Notierung (strukturiertes Umgangsg Englisch, bzw. -deutsch; NIEMEYER, 1975)
- b) Formulierung in einem Pseudo-Assembler (portable Assembler; KNUTH, 1973)
- c) Formulierung in LISP bzw. SNOBOL (ELSON, 1975; bestehende problemorientierte Programmiersprache mit implementierten Compilern)
- d) Formulierung in einer höheren strukturierten Problemsprache (PASCAL; WIRTH, 1975)

Von allen aufgeführten Möglichkeiten scheint uns die unter a) genannte Möglichkeit die praktikabelste zu sein. Wird die allgemeine Strukturformulierung zusätzlich durch einen graphischen Ablaufplan ergänzt (NIEMEYER, 1975), so haben wir Algorithmen, die unmittelbar in Assembler oder eine problemorientierte Sprache zu übertragen sind.

Die unter b) genannte Möglichkeit scheidet aus, da sich die von KNUTH (1973) geforderten Eigenschaften von 'MIXAL', einer hypothetischen Assemblersprache, nur sehr schwer in Eigenschaften einer problemorientierten Sprache überführen lassen (Byte-Operation, Shift-Vorgänge).

In LISP formulierte Algorithmen lassen sich gut in andere Sprachen übertragen.

Die speziell für Programme dieser Arbeit entwickelten Übertragungsregeln von LISP nach FORTRAN seien im folgenden wiedergegeben.

## 2.1 Regeln zur Übertragung von Algorithmen aus LISP in FORTRAN

| <u>LISP-Element</u>      | <u>FORTRAN-Äquivalent</u>                                                                                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| a) IF E = P THEN         | IF(E.EQ.P)...                                                                                                                                                               |
| b) IF E = P THEN ERROR   | RETURN<br>(Vorher muß eine Fehlervariable gesetzt sein, die vor dem letzten Return auf Null gesetzt wird)                                                                   |
| c) IF E = P THEN [     ] | Gruppe von Befehlen. In FIV Abprüfen der verneinten Bedingung. Es wird eine Marke eingeführt.<br>IF(.NOT.(E.EQ.P)) GO TO 1<br><br>1 CONTINUE                                |
| d) DO STEP 2 I TIMES     | DO 2 L = 1,I<br><br>2 CONTINUE                                                                                                                                              |
| e) RETURN (Q)            | Rückgabe von Q (Klammerinhalt muß den Namen eines formalen Rückgabeparameters erhalten) als Subroutine-Parameter, d.h. Schaffung eines neuen Parameters.<br>R = Q<br>RETURN |

- f) DO STEP 2 I TIMES; RETURN Q    Nach Beendigung der Schleife soll Q zurückgegeben werden.
- DO 2 L = 1,I
- 2 CONTINUE
- R = Q
- RETURN
- g) Q:LINK    Zugriff auf Adresse Q, um den Parameter LINK zu lesen.
- IR = Q
- READ (IUNIT'IR)LINK
- h) Q ← S    Zuordnung Q = S
- i) Q ← Q:LINK    Abarbeitung von rechts. Zugriff auf Adresse Q, um LINK zu lesen.
- Zuordnung Q = LINK
- j) Q:LINK ← E:LINK    Abarbeitung von rechts (s.o.)
- Zugriff auf Adresse E, um LINK zu lesen. Zugriff auf Adresse Q, um Variable zu lesen, in die LINK gespeichert werden soll.
- Zuordnung LINKD = LINK
- k) E:LINK1:LINK2    Abarbeitung von links. Zugriff auf E, um LINK1 zu lesen. Zugriff auf LINK1, um LINK2 zu lesen.
- READ (IUNIT'E)LINK1
- READ (IUNIT'LINK1)LINK2

Ein ähnlicher Versuch, eine höhere strukturierte Problemsprache (PASCAL) in FORTRAN zu übertragen, gelingt nicht (Möglichkeit d), da z.B. PASCAL qualitativ andere Sprachelemente enthält als FORTRAN (so z.B. die Definitionsmöglichkeit beliebiger Datentypen (einschließlich Pointer), Rekursion, u.a.).

Dennoch kann versucht werden, strukturierte Programme in FORTRAN zu überführen. Es seien hier die wichtigsten Umformungsregeln wiedergegeben (GIERE, HEGER, KRIER, 1976). Auf die strukturierte Programmierung soll in diesem Zusammenhang allerdings nicht weiter eingegangen werden.

## 2.2 Umsetzung von Strukturblöcken in FORTRAN

FORTRAN ist nicht blockstrukturiert, d.h. man kommt nicht ohne Sprunganweisungen aus und kann keine lokalen Variablen definieren.

Für die Umsetzung müssen daher Hilfskonstruktionen gewählt werden, mit denen Strukturblöcke simuliert werden können. Die dabei verwendeten GO TOs sind keine im Sinne der von DIJKSTRA (1968) verbotenen GO TO.

- a) Einfache Strukturblöcke bedürfen keiner Umwandlung, allenfalls können Anweisungsfolgen als SUBROUTINE (oder als FUNCTION) zusammengefaßt werden. Dies ist allerdings ein speicher- und zeitaufwendiges Vorgehen.
- b) Die einseitige Bedingung der Form IF B THEN S (B für Bedingung; S für Statement) wird wie folgt übertragen:  
IF (B) S.

In allen anderen Fällen, in denen der Strukturblock S nicht aus einer einfachen Anweisung besteht, muß folgendes verwendet werden:

```
      IF (.NOT.B) GO TO 1
      S
1 CONTINUE
```

Dies ist z.B. dann der Fall, wenn S selbst eine IF-THEN-ELSE- oder eine WHILE-DO-Struktur ist.

Die zweiseitige Bedingung IF B THEN S1 ELSE S2 wird wie folgt umgesetzt:

```
      IF (.NOT. B) GO TO 1
      S1
      GO TO 2
1 CONTINUE
      S2
2 CONTINUE
```

c) Der Wiederholungs-Strukturblock WHILE B DO S wird umgesetzt in:

```
1 CONTINUE
      IF (.NOT. B) GO TO 2
      S
      GO TO 1
2 CONTINUE
```

Bei den meisten Schleifenkonstruktionen soll ein Laufindex I von IA bis IE laufen. In diesem Fall ist der Wertebereich vorher zu überprüfen.

```
      I = IA - 1
1 CONTINUE
      IF (I .GE. IE) GO TO 2
      I = I + 1
      S
      GO TO 1
2 CONTINUE
```

oder auch

```
IF (IA .GT. IE) GO TO 2
DO 1 I = IA, IE
S
1 CONTINUE
2 CONTINUE
```

d) Die Fallunterscheidung CASE -OF wird wie folgt übertragen:

```
IF (E .LT. 1 .OR. E .GT. N) GO TO 9
GO TO (1, 2, 3, ..., N), E
1 CONTINUE
S1
GO TO 9
2 CONTINUE
S2
GO TO 9
.
.
.
N CONTINUE
SN
GO TO 9
9 CONTINUE
```

Die aufgeführten Umsetzungsregeln können nicht immer strikt angewendet werden. Für die praktische Programmierung hat sich aus eigener Erfahrung eine weitere Regel ergeben:

Man verwende, wenn möglich, nur Vorwärtssprünge.

Rückwärtssprünge sollen generell auf einige wenige Sammelrücksprungpunkte erfolgen. Hinter diesen kann sich jeweils unmittelbar ein Vorwärtsverteiler befinden.

```
Statt      S1
           2 CONTINUE
           S2
           3 CONTINUE
           S3
           GO TO 3
```

sollte besser folgende Form verwendet werden:

```
1 CONTINUE
  GO TO (2, 3, ..., N), M
2 CONTINUE
  S2
3 CONTINUE
  S3
  .
  .
  .
  GO TO 1
```

Im Hinblick auf die Erstellung eines 'Tool-Pools' wird aber folgender Weg vorgeschlagen:

- a) Darstellungssprache eines Algorithmus  
(a-d)
- b) Allgemeine strukturierte Notierung  
( 'Standard' )
- c) Realisierungssprache

Dies bedeutet, daß in der Regel keine direkte Transformation aus einer Darstellungssprache in eine Realisierungssprache erfolgen sollte, obwohl dies (wie oben gezeigt) möglich und in manchen Fällen auch praktikabel ist. Würde man diesen Weg einschlagen, wäre von jeder

der  $n$  Darstellungssprachen zu jeder der  $m$  Realisierungssprachen je eine Übertragungsprozedur (Regelwerk) notwendig (insgesamt  $n \times m$ ). Wird hingegen grundsätzlich eine Systemkomponente in eine allgemeine strukturierte Notierung überführt, so reduziert sich die Anzahl der Übertragungsprozeduren auf  $n + m$ .

Dieses Prinzip ist z.B. die Grundlage für einen standardisierten Programmaustausch (Standardisierung von Programmsprachen, FRIEDRICH, 1975).

Für den Entwurf von Systemen und deren Komponenten ist ein strukturiertes Vorgehen zu fordern (Schritt b und c; vgl. RSII). Nur so kann ein Programm (-system) den allgemeinen Gütekriterien entsprechen:

- einfach zu testen
- leicht adaptierbar
- benutzerfreundlich
- fehlerfrei
- wartbar.

Für die im ersten Teil dieser Arbeit aufgeführten Such- und Änderungsverfahren wurden nach dem obigen Schema (im Rahmen der Lehrtätigkeit an der Staatlichen Lehranstalt für Medizinische Dokumentations-Assistenten) zahlreiche kleinere Demonstrationsprogramme in Standard FORTRAN erstellt (sie behandelten u.a. Operationen in verketteten Beständen, Binäre Suche, Hash-Verfahren, Invertierung von Datenbeständen, Operationen mit Binärstrings, Operationen an Binär-Bäumen, Operationen an Vielweg-Bäumen), auf deren Wiedergabe im Rahmen dieser Arbeit allerdings verzichtet wird zugunsten der Darstellung von drei umfassenden Systemen (REMED, RS-II, IAS), in denen wir Datenbankorganisationsprinzipien exemplarisch auf kleineren EDV-Anlagen realisiert haben. Ein stark erweiterter 'Tool-Pool' soll an anderer Stelle veröffentlicht werden.

### 3 Interaktion mit einer Datenbank

Für den Benutzer sind vor allem zwei Gebiete bezüglich der Interaktion mit einer Datenbank interessant:

1. Die Eingabe von Daten in die Datenbank
2. Die Abfrage von Daten unter bestimmten Bedingungen

#### 3.1 Eingabe von Daten in die Datenbank

Dem Benutzer muß eine Beschreibung der in der Datenbank gespeicherten Daten zu Verfügung gestellt werden. Eine solche Beschreibung bezeichnet man als "data description table", sie enthält Beschreibungen der Daten bis hinunter zur Feldebene (hier z.B. Name, Länge und Attribute der Felder). Daneben kann eine solche Tabelle auch Angaben über die Zugriffsmöglichkeiten zu den Datenelementen enthalten. Die Datenbeschreibungstabelle (= "Map") wird getrennt von den Daten gespeichert und stellt ein Bindeglied zwischen Benutzer und Datenbank dar, indem sie für den Benutzer (Abfragesprache oder Rechenmodul) alle Attribute von Feldern liefert. Für die Datenbank selber sind diese Angaben für den Aufbau der Speicherstruktur nötig.\*

Eine einfache Eingabe von Daten in eine Datenbank kann z.B. über Terminal (Bildschirm) vorgenommen werden. Die Führung des Benutzers übernimmt hierbei das sogenannte "Format" für die Eingabe.

Um nicht bei jeder Datenerfassung diese Formate neu programmieren zu müssen und um diese Formate flexibel zu halten, wird man sich zweckmäßigerweise eines "Formatgenerators" bedienen. Das Konzept eines Formatgenerators

---

\*Eine Beschreibung der Datenbank könnte auch durch eine Data-Description-Language DDL vorgenommen werden.

ist an anderer Stelle ausführlich beschrieben worden.  
(SAGER, 1976)

Das Kernstück eines solchen Programms ist die oben angesprochene Datenbeschreibungstabelle, die neben den Feldnamen und den Feldattributen auch noch Angaben über anzuwendende Prüfroutinen beinhalten muß. Im Anhang ist die Realisierung eines einfachen "Formatgenerators" in FORTRAN wiedergegeben (DEP-Programm). Sein grundsätzlicher Aufbau sei hier kurz beschrieben. Das Programm-Konzept geht davon aus, daß I/O auf Bildschirm möglich ist. Es wird auf die Möglichkeit der Cursor-Positionierung zwangsläufig verzichtet. Für die Beschreibung der Felder wird eine 'Map' definiert, die wie folgt aufgebaut ist:

Jedes Feldelement wird durch eine Zeile beschrieben. Sie enthält den Namen des Feldes, eine Kennzeichnung des Feldes, eine Kennzeichnung des Datentyps (alpha, numerisch), eine Angabe der Feldlänge (Positionen im logischen Satz), sowie eine Angabe über die zugelassenen Prüfroutinen für dieses Feld (=Marke in einem Prüfprogramm).

| <u>Name</u> | <u>Typ</u>  | <u>Länge</u> | <u>Prüfroutine</u> |
|-------------|-------------|--------------|--------------------|
| NAME        | 0           | 17-47        | 3                  |
|             | (alpha)     |              |                    |
| GESCHL      | 1           | 48           | 1                  |
|             | (numerisch) |              |                    |
| :           | :           | :            | :                  |

Nachdem im Dialog am Bildschirm eine Anmeldung des Benutzers für eine Datenbank erfolgt ist, wird jeweils ein Führungstext aus der 'Map' ausgegeben. Die darauf erfolgende Eingabe wird nach Typ und Länge formal geprüft,

sowie nach der Prüfroutine, die in einem getrennten Prüfmodul aufgerufen wird, auf logische Bedingung abgefragt. Bei auftretenden Fehlern erscheint der Führungstext noch einmal, und es wird eine erneute Eingabe des gleichen Feldes erwartet.

Nach Abarbeitung der 'Map' wird dem Benutzer das gefüllte Format vorgespielt. Sofortige Korrekturmöglichkeiten sind gegeben. Zu diesem Zweck muß der Name des zu korrigierenden Feldes und der betreffende Feldinhalt eingegeben werden. Erst nach Quittierung des gesamten Formats werden die eingegebenen Daten als logische Sätze in die Datenbank übergeben.

### 3.2 Abfrage von Daten

Dem Benutzer stehen zur Abfrage von Daten nach logischen Bedingungen folgende Möglichkeiten offen: Er kann sich einer Programmiersprache, einer erweiterten Programmiersprache<sup>\*</sup> oder einer Abfragesprache bedienen. Bei der letzten Form, der eigentlich interessanten für den Dialog, ist weiterhin eine Unterscheidung zwischen compilierter und interpretierter Sprache zu treffen.

Eine Abfragesprache muß folgende Elemente enthalten: Sie muß als Operanden Feldname und Feldinhalte unterscheiden, als Operatoren logische Verknüpfungsoperatoren, sowie Vergleichsoperatoren und arithmetische Funktionen anbieten.

Wir haben im Rahmen der Realisierung von Recherchemodulen den Weg eines Interpreters für logische und arithmetische FORTRAN-Ausdrücke beschritten.

---

<sup>\*</sup>Data-Manipulation Language (DML)

Der Interpreter kann auf einen usual file ebenso wie auf einen inverted file angewendet werden.

Die allgemeinen Prinzipien der Formelübersetzung seien hier dargestellt (BERZTISS, 1975). Weiter unten befindet sich eine Beschreibung der Realisierung eines Interpreters im Rahmen der Darstellung von REMED.

Eine eindeutige Schreibweise von logischen und arithmetischen Ausdrücken ohne Benutzung von Klammern ist von LUKAZIEWICZ entwickelt worden. Man nennt diese Schreibweise "Polnische Notation" oder post fix-Notation.

Der Grundgedanke dabei ist, den Operator hinter die beiden Operanden zu schreiben, die er verknüpft.  $A+B$  wird zu  $AB+$ . Aus  $A+B*C$  ergibt sich  $ABC*+$ . Bei der Auflösung werden die beiden Operanden, auf die ein Operator folgt, miteinander durch diesen Operator verknüpft (hier:  $BC*$ ). Das Ergebnis (E) wird an die Stelle von  $BC*$  gesetzt. Jetzt kann die nächste Operation  $AE+$  ausgeführt werden.

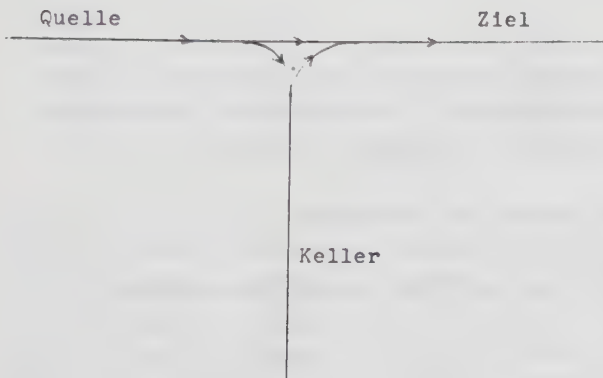
$(A+B)*C$  wird zu  $AB+C*$ . Hier wird erst addiert und dann das Ergebniss mit C multipliziert. Am letzten Beispiel sieht man, dass Klammern nicht benötigt werden.

Zur Umwandlung eines logischen oder arithmetischen Ausdrucks aus der mathematischen Schreibweise in die "Polnische Notation" wird ein Keller oder Stack benötigt. Dies ist ein Speicher mit der Eigenart, daß nur das jeweils zuletzt abgespeicherte Element als erstes wieder greifbar ist. (last in - first out = LIFO)

Wird ein Element in den Keller gebracht, so werden alle anderen Elemente, die bereits im Keller vorhanden sind, um eine Position nach "unten" gestoßen. Das bisher 1.

Element wird zum 2., das  $n$ -te zum  $n+1$ ten. Man nennt diesen Vorgang BRINGEN oder Push down. Wird ein Element aus dem Keller geholt, so werden alle anderen Elemente, die noch im Keller vorhanden sind, um eine Position nach oben angehoben. Das bisher 2. Element wird zum 1., das  $n$ -te wird zum  $n-1$ ten. Man nennt diesen Vorgang HOLEN oder Pop up.

Wie ein Keller oder Stack bei der Formelübersetzung benutzt wird, kann vielleicht durch folgende Skizze veranschaulicht werden:



Dabei ist die "Quelle" ein Speicher, der einen arithmetischen oder logischen Ausdruck in der "mathematischen" Schreibweise (bestehend aus einer Folge von Operanden und Operatoren und Klammern) als Inhalt hat.

"Ziel" ist ein Speicher, der den in eine "Polnische Notation" umgewandelten Ausdruck aufnimmt.

Soll ein Element aus der Quelle ins Ziel transportiert werden, so kann dies entweder direkt oder indirekt (über eine Warteposition im Keller) geschehen.

Der Übersetzungsvorgang geht im Einzelnen wie folgt vor sich:

1. Operanden werden direkt von der Quelle ins Ziel transportiert. Operatoren nehmen immer den indirekten Weg, dabei spielt allerdings der Rang des Operators eine Rolle (\* und/oder haben einen höheren Rang als + und -).
2. Steht bei der Abarbeitung der Quelle ein Operator an, dessen Rang größer ist als der des 1. Keller-Elementes (Operator), so wird dieser Quell-Operator in den Keller gebracht (Push-down).

Beispiel:  $A-B+C$  wird zu  $AB-C+$

| Zustand-Nr. | Quelle | Keller | Ziel  |
|-------------|--------|--------|-------|
| 1           | A      | leer   | leer  |
| 2           | -      | leer   | A     |
| 3           | B      | -      | A     |
| 4           | +      | -      | AB    |
| 5           | +      | leer   | AB-   |
| 6           | C      | +      | AB-   |
| 7           | leer   | +      | AB-C  |
| 8           | leer   | leer   | AB-C+ |

3. Ist der Rang des Quell-Operators kleiner oder gleich dem Rang des obersten Kellerelementes (Operators), so bleibt der Quelloperator in seiner Position. Der 1. Keller-Operator wird aus dem Keller geholt (Pop-up) und ins Ziel transportiert.

Beispiel:  $A+B \times C$  wird zu  $ABC \times +$

| Zustand-Nr. | Quelle   | Keller     | Ziel           |
|-------------|----------|------------|----------------|
| 1           | A        | leer       | leer           |
| 2           | +        | leer       | A              |
| 3           | B        | +          | A              |
| 4           | $\times$ | +          | AB             |
| 5           | C        | $\times +$ | AB             |
| 6           | leer     | $\times +$ | ABC            |
| 7           | leer     | +          | $ABC \times$   |
| 8           | leer     | leer       | $ABC \times +$ |

Der Zustand "leer" in der Quelle und im Keller hat die gleiche Wirkung wie ein Operator mit dem niedrigsten Rang (vgl. aber Regel 6 und 7).

Steht in der Quelle ein Ausdruck mit Klammern an, so müssen weitere Regeln beim Übersetzungsvorgang beachtet werden.

4. Eine öffnende Klammer wird grundsätzlich in den Keller gebracht.
5. Steht bei der Abarbeitung der Quelle eine schließende Klammer an und als 1. Kellerelement eine öffnende Klammer, so wird diese aus dem Keller geholt. Öffnende und schließende Klammern werden dann gelöscht. (Ist das erste Kellerelement nicht eine öffnende Klammer, so wird nach Regel 3 verfahren.)

Beispiel:  $A*(B-C)$  wird zu  $ABC-*$

| Zustand-Nr. | Quelle | Keller | Ziel  |
|-------------|--------|--------|-------|
| 1           | A      | leer   | leer  |
| 2           | *      | leer   | A     |
| 3           | (      | *      | A     |
| 4           | B      | (*     | A     |
| 5           | -      | (*     | AB    |
| 6           | C      | -(*    | AB    |
| 7           | )      | -(*    | ABC   |
| 8           | )      | (*     | ABC-  |
| 9           | leer   | *      | ABC-  |
| 10          | leer   | leer   | ABC-* |

Zur Vervollständigung der Regel seien noch die Vorgehensweisen für den Abschluß einer Übersetzung angegeben:

6. Ist die "Quelle" leer, so werden die im Keller noch vorhandenen Operatoren ins Ziel überführt.
7. Ist der Keller leer, so wird der Quell-Operator im Keller abgelegt.

Durch eine Verallgemeinerung der Hierarchie der Operatoren kann man mit dieser Methode unter anderem die Zuweisung (Anweisung), Aufruf von Funktionen, die BOOLEsche Arithmetik u.a. übersetzen.

Die verallgemeinerte Hierarchie lautet:

- |                    |                     |
|--------------------|---------------------|
| 1. (               | öffnende Klammer    |
| 2. )               | schließende Klammer |
| 3. =               | Zuordnungszeichen   |
| 4. .OR. bzw. .XOR. | logisches Oder      |
| 5. .AND.           | logisches Und       |
| 6. .NOT.           | logische Negation   |

- |                     |                                    |
|---------------------|------------------------------------|
| 7. +-               | Addition, Subtraktion              |
| 8. */               | Multiplikation, Division           |
| 9. **               | Potenzierung                       |
| 10. +/-             | positives und negatives Vorzeichen |
| 11. Funktionsaufruf |                                    |

Interessanterweise kann man zur Erzeugung einer Befehlsfolge aus dem "Polnischen String" wiederum das gleiche Verfahren, wie vorher beschrieben, verwenden.

Der "Polnische String" stellt nun die Quelle dar und wird von links nach rechts Element für Element abgearbeitet.

Es gelten dabei folgende Regeln:

1. Ist das Element ein (Verweis auf einen) Operand, so wird dieser in den Keller gebracht.
2. Ist das Element ein (binärer) Operator, so werden die beiden oberen Elemente (=Operanden) (bei unären Operatoren das oberste Element) aus dem Keller geholt. Auf sie (es) wird der Operator angewandt, und das Ergebnis wird wieder in den Keller gebracht, es sei denn, der Operator war das Zuordnungszeichen (=).

Damit ist eine Möglichkeit der Formelübersetzung im Prinzip beschrieben. Führt man nun Formel für Formel die Schritte einzeln durch, so führt man eine sequentielle Formelübersetzung aus, nach dem Prinzip eines "Interpreters".

Führt man für eine Reihe von Formeln zunächst nur immer die ersten Schritte durch, und dann gesondert für alle den Schritt der Erzeugung einer Befehlsfolge, so geht man nach dem "Compiler-Compiler"-Prinzip vor.

4 Darstellung von drei Systemen als Beispiele zur Realisierung von Datenbankorganisationsprinzipien auf kleineren EDV-Anlagen

4.1 Recherche-Moduln zur Erschließung medizinischer Daten, REMED

REMED ist eine Zusammenfassung von Moduln, die zur Nutzbarmachung und Aktivierung medizinischer Datenbestände (z.B. EKG-Analysen und Krankenblattdokumentation) dienen.

Die Suchanfragen vom Typ "welche Patienten ..." können On-Line im Dialog gestellt werden. Die in der Form von arithmetischen und/oder booleschen Anweisungen gestellten Fragen erlauben sämtliche Operatoren, die in FORTRAN vorgesehen sind (einschließlich Vergleichsoperationen und Funktionen). Befehle werden durch einen Formelinterpreter abgearbeitet. Es kann auf eine Datei mit festen und/oder variablen Satzlängen, sowie auf eine teilverteilte Datei zugegriffen werden.

4.1.1 Mögliche Datenstrukturen in REMED

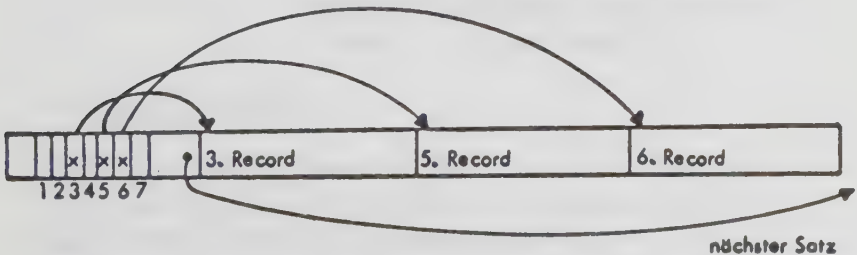
a) Usual File

BABAD (1977) hat die möglichen Formen des Aufbaus von Sätzen zusammengestellt:

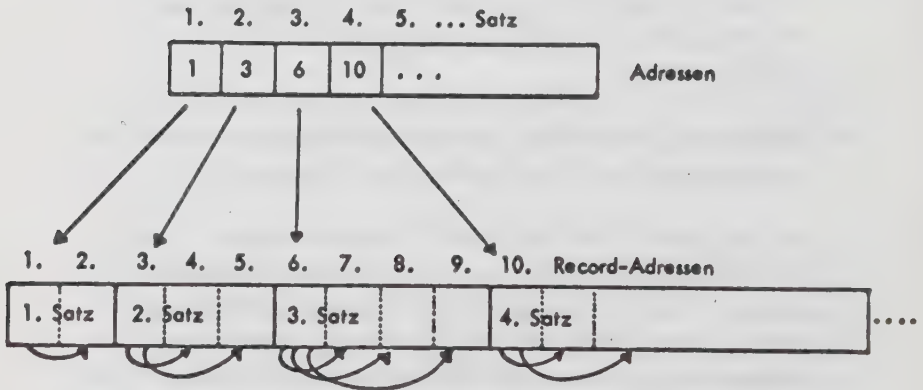
(1) "Positional": Feste Länge der Felder.

- (2) "Relational": Variable Länge der Felder; alle Felder müssen vorhanden sein; zur Abgrenzung wird ein spezielles Symbol verwendet.
- (3) "Indexed": Ein Record wird durch einen Satz von Pointern erweitert, die auf Anfang oder Ende von Feldern weisen. Es müssen nicht alle Felder vorhanden sein.
- (4) "Labeled": Jedes Feld wird durch einen bestimmten Label identifiziert. Felder können fehlen.
- (5) "Fixed Length with Overflow": Felder werden in Teile fester Länge und in Teile variabler Länge aufgeteilt. Paßt ein Feld in einen Teil fester Länge hinein, wird die "Positional"-Methode verwendet, anderenfalls werden der "überstehende Teil" nach der Methode "Indexed" gespeichert und ein Kontroll-Bit gesetzt.

In REMED wird für einen Satz eine modifizierte "Indexed"-Form verwendet. Ein Satz kann sich aus 2 bis 13 Records zusammensetzen, hat also variable Länge. Einem Kopfsatz können 1 bis 12 Records folgen, die durch Vorwärtsverknüpfung mittels einer Anwesenheitsliste miteinander verbunden sind. (Auf diese Weise wird ein erheblicher Speicherplatz gegenüber fester Satzlänge eingespart.)



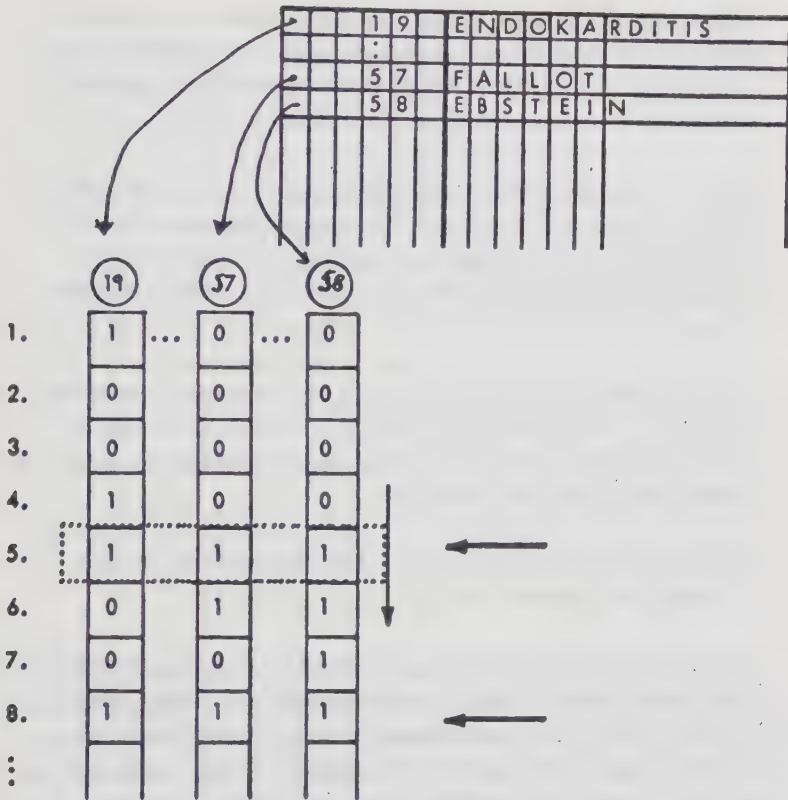
Außerdem wird aus dem Kopfsatz noch auf den nächsten Satz gepointet. Der Randomzugriff läuft über eine Verweisdatei ("Index").



b) Inverted File

Schnelle Recherchen werden auf der Basis der teilinvertierten Dateien durchgeführt. Für die wichtigsten Merkmale (bzw. Merkmalsausprägungen) werden invertierte Files in Bitstrings angelegt. Jeder Bitstring ist über ein Lexikon mit einem FORTRAN-Namen anzusprechen.

Beispiel: Geschlecht : SEX  
Katheter links : KATHLI  
Alter über 60 : ALT60



Die Positionen im Bit-String sind als Ordinalzahlen, die laufenden Nummern der Sätze. Ein Zugriff zu einem Satz variabler Länge läuft dann wieder über die Verweisdatei. Die Auswertung über die Bitstrings erfolgt dann nach Verfahren, wie sie SELBMANN (1974) beschrieben hat.

#### 4.1.2 Abfragesprache

Besonderes Gewicht wurde in REMED auf die Implementierung einer Abfragesprache gelegt. Im Rahmen des Programm-Systems wurde ein Formelinterpretierer für FORTRAN-Anweisungen erstellt. Er bietet folgende Vorteile:

- Es braucht keine neue Rechtersprache erlernt zu werden. Rechercheausdrücke werden in FORTRAN formuliert, d.h. praktisch kann jeder ohne längere Einweisung eine Recherche durchführen.
- Sowohl für eine Recherche in der Database als auch für die Invertierung von Merkmalen als auch für die Recherche in der invertierten Datei werden gleiche Anweisungen (in FORTRAN) benutzt. Alle drei Vorgänge können on-line durchgeführt werden.
- Keine Einschränkungen bei der Formulierung von Suchanfragen bezüglich Komplexität und der Verwendung von arithmetischen und booleschen Operatoren und Funktionen sind zu beachten.
- Die Ausgabelisten, die auch neu errechnete Größen beinhalten dürfen, sind frei zu gestalten.

Es stellt sich das praktische Problem, wie nach FORTRAN-Regeln geschriebene Anweisungen, die einem FIV-Programm als Daten angeboten werden, von diesem "verstanden" und so interpretiert werden können, daß als Ergebnis genau der gleiche Wert vorliegt, als wenn diese Anweisung Teil eines compilierten und gebundenen FIV-Programms wäre.

Die Prinzipien des in REMED realisierten Interpreters seien im folgenden wiedergegeben.

Zunächst müssen aus einem String von FIV-Zeichen Operatoren (arithmetische und boolesche) Funktionsaufrufe,

Variablen, Konstanten in ihren verschiedenen Darstellungsarten identifiziert werden.

#### a) Stringzerlegung

Man führt eine Stringzerlegung in Teilstrings durch. Als Begrenzer (Delimiter) von Teilstrings werden dabei alle FIV-Sonderzeichen bzw. das abschließende Blank angesehen. Jeder Teilstring  $i$  der Länge  $n$  füllt dabei die ersten  $n$  Positionen einer Zeile der Länge  $m$  einer Matrix; der Delimiter ist jeweils die  $i$ -te Position eines Vektors. Ein Teilstring kann dabei auch leer sein.

Beispiel:  $X = AB * (CDEF - BYK)$

|      |   |
|------|---|
| X    | = |
| AB   | * |
|      | ( |
| CDEF | - |
| BYK  | ) |

Eingestreute Blanks werden dabei von vornherein eliminiert. In der Praxis wird man zweckmäßigerweise am Anfang allen Zeichen eine Ordinal-Zahl zuordnen und dann nur noch mit diesen Zahlen operieren. Aus Anschauungsgründen wird aber hier die Zeichendarstellung beibehalten. Damit befinden sich alle Zeichen für arithmetische Operatoren, sowie Punkte und Kommata in dem Vektor, alle Variablennamen in den Zeilen der Matrix. In den Zeilen befinden sich außerdem noch numerische (Teil-) Strings von Konstanten, sowie die Funktionsaufrufe, genauer die Strings der symbolischen Namen.

#### b) Lexikalische Analyse

Ziel der weiteren Bearbeitung der obigen Matrix und des Vektors ist nun, alle Operanden und Operatoren in Zahlen

umzuformen, die Adressen repräsentieren. Bei den Operanden sind dies Adressen in einem Wertevektor, in dessen Positionen der Wert der betreffenden Variablen (Konstanten) gespeichert ist. Bei den Operatoren sind dies Marken zu Sprungzielen, wo die in Frage stehenden Operationen (Funktionsaufrufe) aufgeführt werden können.

Wir betrachten zunächst Anweisungen mit einfachen Variablen.

Die laufenden Nummern der arithmetischen Operatoren, die nur durch ein Zeichen symbolisiert werden, sind im Vergleich des Begrenzer-Vektors mit einem Operatoren-Vektor leicht festzustellen. Namen von logischen Operatoren und Funktionsaufrufe sind wie Variablennamen durch geeignete Tabellensuchverfahren zu identifizieren.

Eine Variablentabelle ist im Prinzip wie die oben eingeführte Teilstringtabelle aufgebaut. Die Besetzung der Zeilen kann nach verschiedenen Verfahren vonstatten gehen:

- Fortlaufende Eintragung ohne Ordnungsprinzip in der Reihenfolge des Auftretens.
- Alphabetische Eintragung. Bei diesem Verfahren wird entweder ein großer Speicherraum benötigt, oder es wird ständig eine Reorganisation der Tabelle (Sortieren) nötig.

- Eine der vielen Varianten der HASH-Speicherung. Durch einen geeigneten Algorithmus soll eine Gleichbesetzung der Matrix gewährleistet werden. Man bildet z.B. die Summe der Wertigkeiten des ersten und letzten Zeichens und betrachtet das Ergebnis als Zeilenadresse der Tabelle. Ist die Zeile schon besetzt, so muß der "Überlaufbereich" angesteuert werden an einer Stelle, an der nun ein Variablenname abgesetzt werden kann.

An einem einfachen Beispiel soll das Prinzip der Umformung erläutert werden.

Gegeben seien: ein Vektor, der die arithmetischen Operatoren beinhaltet,

eine Variablentabelle, in der die in Frage kommenden Variablennamen verzeichnet sind.

Unsere Matrix und der Begrenzervektor würden dann wie folgt umgeformt werden können:

|      |   |    |   |
|------|---|----|---|
| X    | = | 14 | 3 |
| AB   | x | 11 | 6 |
|      | ( |    | 1 |
| CDEF | - | 13 | 5 |
| BYK  | ) | 12 | 2 |

Zur Unterscheidung zwischen Adressen im Wertevektor und Marken (Operatoren) können letztere z.B. mit negativen Vorzeichen versehen werden. Schreibt man nun die so erhaltenen Zahlen fortlaufend, erhält man einen Zahlenstring.

14 -3 11 -6 -1 13 -5 12 -2

c) Bilden eines Polnischen Strings

Dies ist nun der String, der in der Quelle angeboten wird und über einen Stack in einen polnischen String, wie oben beschrieben, umgewandelt wird.

Um diese Umwandlung vornehmen zu können, muß auch der Rangwert jedes Operators gespeichert sein.

d) Überführung in eine Befehlsfolge

Wie die Überführung eines Polnischen Strings in eine Befehlsfolge abläuft, soll der folgende Abschnitt verdeutlichen.

In der Quelle steht der Polnische String zur Verfügung. Das Programm, das diesen in eine Befehlsfolge umformt, leistet folgendes: Es arbeitet den Quellenvektor Element für Element ab.

1. Ist das nächste Element in der Quelle ein Operand, d.h. hier eine Adresse, unter der in einem Wertevektor der betreffende Wert gespeichert ist, so wird diese Adresse in den Keller gebracht.
2. Ist das nächste Element in der Quelle ein Operator, d.h. eine Sprungmarke (in unserem Beispiel mit negativem Vorzeichen versehen), so wird
  - a) festgestellt, ob der Operator binär oder unär ist, und
  - b) werden davon abhängig zwei oder ein Element aus dem Keller geholt. Da diese Elemente Adressen in einem Wertevektor sind, können die Werte aus diesem Vektor zwei Arbeitsvariablen  $a_1$  und  $a_2$  zugeordnet werden.

- c) Anschließend wird zu der von der Sprungmarke bezeichneten Stelle gesprungen, dort werden die Werte miteinander verknüpft (addiert, multipliziert, u.ä.), das resultierende Ergebnis wird einer weiteren Variablen zugeordnet.
- d) Für den Ergebniswert muß nun ein Platz im Werte-Vektor bereitgestellt werden, d.h. es muß eine Adresse zur Verfügung stehen.
- e) Der Ergebniswert wird an der betreffenden Position des Wertevektors gespeichert.
- f) Die Adresse hiervon wird in den Keller (zurück) gebracht. Die Abarbeitung kann dann bei 1. fortgeführt werden.

Am Ende steht dann die Adresse des Ergebnisses am Eingang des Kellers.

| Zustand-Nr. | Quelle | Keller      | Operation                                                        |
|-------------|--------|-------------|------------------------------------------------------------------|
| 1           |        | leer        | push                                                             |
| 2           | 11     | 14          | push                                                             |
| 3           | 13     | 11,14       | push                                                             |
| 4           | 12     | 13,11,14    | push                                                             |
| 5           | -5     | 12,13,11,14 | pop,pop,Operation (Subtrakt.b.Marke 5), push Ergebnis als Adr.15 |
| 6           | -6     | 15,11,14    | Analog Schritt 5                                                 |
| 7           | -3     | 16,14       | Analog Schritt 5                                                 |
| 8           | leer   | 17          |                                                                  |

Im Werte-Vektor steht nun an der 17. Position das Ergebnis. Dieser Interpreter ist im REMED durch mehrere Unterprogramme in Standard-FORTRAN IV realisiert worden (vgl. Programmanhang).

#### 4.1.3 Beispiele für die Interaktion mit der Datenbank

##### a) Formulierung eines Auswahlausdruckes für 'Vorauswahl' (im 'Inverted File')

Zur Vorauswahl von Sätzen kann ein Auswahlausdruck formuliert werden, der formal wie eine logische IF-Anweisung in FORTRAN aufgebaut ist. Operanden dürfen dabei "logische Variablen" sein, d.h. Namen von invertierten Merkmalen, die in einem Lexikon eingetragen sein müssen.

Beispiel für einen Auswahlausdruck:

```
IF (( R_ERKRANKUNG .OR. ENDOKARDITIS) .AND. T_INSUFFIZIENZ) SELECT RECORD
```

Das Ergebnis der Vorauswahl wird als ERGEBNIS-Bit-String in die teilinvertierte Datei abgestellt. Auf diesen Ergebnis-String kann dann in der eigentlichen Recherche zurückgegriffen werden.

##### b) Formulierung von Suchanfragen für die Recherche (im 'Usual File')

Suchanfragen werden in Form von arithmetischen oder booleschen FORTRAN IV-Anweisungen gestellt. Diese Anweisungen sind nur nach FORTRAN-Regeln aufgebaut.

Beispiele für Anweisungen:

```
KILO = GEWICHT/1000  
M1   = KILO.GT.40  
M2   = (ALTER.EQ.9.OR.ALTER.EQ.10).AND.M1
```

Die verwendeten Variablen müssen dabei in einem Lexikon für Variablen eingetragen sein.

Ein Zugriff auf die Elemente eines Satzes ist außer über Lexikonvariablen auch durch direkte Recordpositions-adressierung möglich (Dollarvariable).

Beispiel: \$01019 bezeichnet das 19. Element des ersten Records eines Patientensatzes.

### c) Anforderung von Output-Listen

Die Anforderung von Output-Listen erfolgt in REMED in einem weiteren Schritt:

Von jedem Patientensatz, der ein logisches Merkmal erfüllt, kann nun eine freigestaltete Liste ausgegeben werden. Dabei können neuberechnete Größen ebenfalls mit ausgegeben werden.

Beispiel: für die Anforderung von Ausgabelisten:

ALTER, GEWICHT, \$0108 - \$01041  
\$01006 - \$01020, KATHLI, KILO

Der Bindestrich symbolisiert hier die Erstreckung 'von-bis'. Falls die Ausgabe auf den Drucker gewünscht wird, kann das Format frei gewählt werden. Wird kein Format angegeben, erfolgt die Ausgabe auf Band binär.

## 4.2 Interaktives Auswertungs-System (IAS)

### 4.2.1 Allgemeines

Bei der Durchführung von statistischen Beratungen wird immer wieder festgestellt, daß nicht die theoretischen Probleme Schwierigkeiten machen und die meiste Zeit beanspruchen (die bereits durch Beratung im Stadium der Versuchsplanung durch den Statistiker behoben werden könnten), sondern vielmehr die pragmatischen Probleme der Datenerfassung, Datenvorbereitung und Häufigkeitsauswertungen. Diese Probleme wurden zunächst durch die Erstellung eines Batch-Programm-Paketes gelöst.

Dennoch sind bei Arbeiten, die mehrere aufeinander aufbauende Auswerteschritte erfordern, lange Bearbeitungszeiten unvermeidbar, da im Batchbetrieb die Erstellung der Parameter- und Steuerkarten langwierig ist und häufig zu Fehlern führt.

Es wurde daher ein Systemkonzept für ein "Interaktives Auswertungssystem" statistischer Daten entwickelt. Das zu erstellende System sollte

- a) portabel (auch für kleinere Anlagen)
- b) benutzerfreundlich und
- c) offen sein.

Diese drei Forderungen wurden wie folgt erfüllt.

- a) IAS ist einschließlich der Datenverwaltung in Standard-FORTRAN geschrieben und vollkommen kompatibel. IAS ist auf einer Minimalkonfiguration von 8 K, Teletype (Display-) Ein- und Ausgabemöglichkeit und drei logisch sequentiellen externen Speichern lauffähig.

- b) IAS bietet eine Reihe von Daten-Manipulationsmöglichkeiten, realisiert durch einen Interpreter, um Daten einem bestimmten Programm-Modul anzupassen. IAS arbeitet mit symbolischen, also weder logischen noch physikalischen Data-Set-Bezeichnungen.
- c) IAS kann ständig durch bestehende statistische Programme, die nur geringfügig geändert werden müssen, erweitert werden. Es besteht somit Datenunabhängigkeit und die Möglichkeit eines modulartigen Aufbaus. IAS bietet außerdem eine Schnittstelle zu anderen Systemen an (DVM-Schnittstelle).

Im folgenden seien die Möglichkeiten von IAS dargestellt.

#### Dateneingabe

Realisiert durch einen 'Formatgenerator' (vergl. S. 121).

#### Datenmodifikation

Als zweiter großer Kreis der Datenvorbereitung ist neben der Strukturänderung die Modifikation der Datenpunkte (values) anzusehen. Hierunter fallen Recording, Transformation, Generierung und Ausführung arithmetischer Ausdrücke, jeweils unbedingt und bedingt (transgeneration cards im BMD, wesentlich verbessert im BMDP; recode, compute and if im SPSS).

Generalisiert werden kann eine Datenmodifikation z.B. durch einen Interpreter. Dieser sollte für eine Operation mehrere Arithmetische und Boolesche Anweisungen zulassen und alle in einer Programmiersprache gebräuchlichen Operationen und Funktionsaufrufe anbieten, außerdem arithmetische Vergleiche von Größen (Konstanten), Record-Variablen und neu gebildeten Variablen erlauben. In Abhängigkeit von Datenmodifikationen können nun Strukturänderungen vorgenommen werden, z.B. Record-Auswahl oder -Änderung in Abhängigkeit von einer Bedingung.

Im Einzelnen wurden folgende Programme realisiert:

- Formatfreie Eingabe
- Korrektur von Einzeldaten
- Transformierung von Variablen, Generierung von Variablen, Missing-Data-Behandlung
- Einfügen oder Löschen von Daten in Sätzen bzw. von Sätzen in Dateien
- Gruppierungsmöglichkeit (Aufteilen und Zusammenschließen von Dateien)
- Bedingte und unbedingte Auswahlmöglichkeiten von Daten aus Sätzen und von Sätzen aus Dateien
- Datenstrukturänderungen (Änderung der Anordnung innerhalb von Sätzen, Änderung der Anordnung der Sätze innerhalb von Dateien, Transponierung mehrdimensionaler Datenstrukturen)
- Mehrdimensionale Output-Möglichkeiten von Datei-Inhalten

#### Datascreening

- Bedingte und unbedingte einfache Häufigkeitsauszählung
- Bedingte und unbedingte mehrdimensionale Häufigkeitsauszählung (Kontingenztafelbewertungen)
- Grundstatistiken (Lage- und Streuungsparameter)

## Inferenzstatistik

- Einfache Zwei-Stichproben-Teste
- Einfache, multiple und polynomiale Regression (mit der Möglichkeit der Plotterausgabe)
- Verschiedene Varianzanalyse-Modelle

Bei der Anwendung des Interaktiven Auswertungs-Systems hat sich gezeigt, daß bei guter Vorausplanung der Auswertungsschritte die erforderlichen Datenmanipulationen und statistischen Berechnungen in wenigen Dialogsitzungen bewältigt werden können.

### 4.2.2 Datenstrukturen von statistischen Daten

Es soll gezeigt werden, daß sich die Datenstrukturen von statistischen Daten durch ein Modell darstellen lassen und daß dieses Modell durch nicht zu umfangreiche Programme darstellbar ist.

#### a) Überlegungen zur Strukturänderung (FRIEDRICH, 1976)

Die Art der Zusammengehörigkeit einer Menge von Datenpunkten kann durch verschiedene (gleichwertige) Darstellungsformen beschrieben werden (Menge von n-Tupeln, Baumdarstellung, k-dimensionale Matrix).

Für statistische Anwendungen erscheint uns die k-dimensionale rechteckige Datenmatrix geeignet zu sein. Sonderfälle hiervon sind einmal die vollbesetzte Matrix und dann die zweidimensionale Matrix, wie sie z.B. SPSS

als generelle Datenstruktur vorstellt. Sie wird aufgespannt durch "cases" mal "variables". In der Tat ist dieser Sonderfall auch die meistgebrauchte Datenstruktur.

Eine Datenmatrix besteht aus folgenden Elementen verschiedener Stufe: Datenpunkt, mehrere linear angeordnete Punkte ergeben eine Dimension, mehrere Dimensionen spannen die Matrix auf.

Folgende Operationen allgemeinsten Art lassen sich auf jeder Ebene anwenden:

- Vermehren der Anzahl durch Hinzufügen von neuen Elementen und (ab Dimensionsebene) durch Aufteilung von Elementen in mehrere;
- Vermindern der Anzahl durch Wegnahme von Elementen und (ab Dimensionsebene) durch Vereinigen von mehreren Elementen;
- Umordnen (jeweils innerhalb einer Ebene).

Zu diesen Operationen seien verschiedene Beispiele gegeben:

auf der Ebene der Daten: innerhalb einer Dimension Zufügen einer weiteren Variablen in einem "case";

auf der Ebene der Dimensionen: die Dimension Meßwiederholung wird fallengelassen (es interessiert z.B. nur noch jeweils die Erstmessung);

auf der Ebene der Datenpunkte: innerhalb einer Dimension wird die Reihenfolge der Variablen verändert; es soll z.B. eine abhängige Variable als letzte zu stehen kommen;

auf der Ebene der Dimensionen: die Reihenfolge der Dimensionen wird verändert; statt "Personen mal Bedingungen mal Messungen" wird "Bedingungen mal Personen mal Messungen" gewünscht. Dieses Problem mündet in die Transponierung bzw. in die Sortierung.

#### b) Überlegungen zur Darstellungsänderung

Die Datenstruktur des k-dimensionalen Datenkörpers kann in verschiedenen Darstellungen repräsentiert werden.

- (1) Zunächst kann man einen k-dimensionalen Datenkörper linearisieren. Es bedarf lediglich einer Vereinbarung darüber, welcher Index am schnellsten laufen soll (vgl. Unterschiede FORTRAN - ALGOL bei der Darstellung mehrdimensionaler Felder). Ferner werden Angaben über die Länge (L) der Felder benötigt. Eine Abbildungsfunktion ergibt sich dann bekanntermaßen als

$$a ( l_1, l_2, l_3, \dots, l_n ) = \sum l_i P_i$$

$$P_i = \frac{1-1}{7^L} L_j \text{ für } i > 1; P_1 = 1$$

Bei der Realisierung können alle Datenpunkte, die zur (zu den) letzten m Dimension(en) gehören, auf einem logischen Record abgesetzt werden. Dabei ist die Datenstrukturinformation nach wie vor implizit gegeben.

- (2) Das andere Extrem ist die Darstellung in Vektoren der Länge  $n (=k+1)$ , in denen die ersten  $k$  Elemente Angaben über die Indizes des Datenpunktes (value) enthalten, der selbst in der  $n$ -ten Position steht. Hier wird der Tupel-Vektor der Länge  $n$  als ein logischer Record abgesetzt. Der Datenkörper liefert dann besetzte Datenpunkte mit expliziter Strukturinformation ab. Diese Struktur ist das Verfahren der Wahl bei einer Matrix, die sehr viele missing data aufweist. Bei welcher Besetzungszahl einer Matrix die Speicherungsart (2) zu einer Einsparung von Speicherplatz gegenüber der Speicherungsart (1) führt, ist angebbbar. Diese Form ist z.B. im Programmsystem EDSGN (Analysis of experiments in complete and balanced incomplected block designs) von CZEGLÉDY und LINDER (1975) verwirklicht.
- (3) Viele Schwierigkeiten bei der Dateneingabe für Programme entstehen u.E. aus der Verwendung einer Struktur, die eine Mischform von (1) und (2) darstellt. Hier ist ein Vektor der Länge  $m$  gegeben. Die ersten  $l$  Elemente enthalten dann Indexangaben über die ersten  $l$  Dimensionen. In den restlichen Elementen befinden sich Datenpunkte der letzten Dimension(en) (Vektor bzw. Untermatrix in linearisierter Form), d.h. in einem logischen Record enthalten die Elemente sowohl explizite Datenstrukturinformation als auch implizite Information, wie unter (1) beschrieben (z.B. im Programmsystem ASCOP von COOPER (1969)).

Eine weitere nicht zu unterschätzende Schwierigkeit ergibt sich bei der Eingabe von Daten in den Speicher, d.h. bei der Überführung von externer in interne Darstellung. Bei NELDER (1969) sind die Hauptformen der externen Darstellung in Indexform zusammengestellt. Er unterscheidet chain, merge, cross, nested und triangle. Diese Formen ergeben zwei Vektoren hintereinander, zwei parallele Vektoren, eine Matrix, eine Neststruktur und eine Dreiecksmatrix. Sie lauten im einzelnen:

$(I = 1 \dots M) , (J = 1 \dots N)$   
 $(I, J = 1 \dots N)$   
 $((I = 1 \dots M) , J = 1 \dots N)$   
 $((I = 1 \dots M) , J = 1 \dots N(I))$   
 $((I = 1 \dots N) , J = 1 \dots I)$

Aus diesen Elementen lassen sich komplexere Strukturen zusammenfügen. Es ist zu überlegen, ob diese Schwierigkeiten nicht generell durch eine Sprache, die die Behandlung von Struktur- und Modellformeln bewältigt, anzugehen sind.

c) Realisierung von Struktur- und Darstellungsänderung durch Programme

Es ergeben sich folgende theoretisch mögliche Kombinationen aus Operationen und Darstellungsform: Die Operationen 'Vermehren', 'Vermindern' und 'Umordnen' können jeweils theoretisch kombiniert werden mit der Form k-dimensionale Matrix, n-Tupel-Form oder Mischform.

Nicht jede Form ist gleichermaßen gut geeignet, um leicht jede Operation ausführen zu können. Bei der Realisierung wird daher bei ungünstiger Kombination zwi-

schen Struktur und Operation zuerst die Struktur geändert und dann die Operation angewendet.

Die Operationen 'Vermehren' und 'Vermindern' greifen immer nur an die letzte Dimension, bzw. an die letzten beiden Dimensionen an. Durch Wiederholung dieses Prinzips sind in mehreren Schritten sehr umfangreiche Änderungen zu erzielen.

Auch die Datenmodifikation bezieht sich nur auf die letzte Dimension; für welche Werte der übergeordneten Indizes eine Modifikation vorgenommen werden soll, kann angegeben werden (allgemeine Form einer geschachtelten DO-Schleife, PL/1-Form).

#### (1) Darstellungsänderung

Realisierungsmöglichkeiten der Überführung von einer Form in die andere sind die folgenden:

- Umformen einer Datenmatrix, so daß statt der letzten  $m$  Dimensionen jetzt die Datenpunkte der letzten  $m_1 \times$  Dimensionen in einen logischen Record abgesetzt werden
- Umformen einer Datenmatrix in eine  $n$ -Tupel-Form
- Umformen einer Datenmatrix in eine Mischform
- Umformen einer Tupel-Form in eine Datenmatrix
- kein Umformen notwendig
- Umformen einer Tupel-Form in eine Mischform

- Umformen einer Mischform in eine Datenmatrix
- Umformen einer Mischform in eine Tupel-Form
- Umformen einer Mischform, so daß statt der 1 Indizes und der letzten m Dimensionen jetzt die  $m+1$  Indizes und die Datenpunkte der letzten  $m+1$  Dimensionen in einem logischen Record abgesetzt werden.

## (2) Strukturänderung

Es seien  $z, y, x$  die Dimensionen des Datenkörpers,  $l_z, l_y, l_x$  die Längen der Dimensionen,  $z_k, y_j, x_i$  aktuelle Indexwerte der Dimensionen. Ein Punkt symbolisiert das Laufen eines Indexes.

- Vermehren durch Hinzufügen

Ein vorhandener Datenkörper wird mindestens noch einmal wiederholt. Bei jedem Indextupel des vorhandenen Datenkörpers wird zunächst eine Indexposition angehängt, die für alle Tupel einen aktuellen Indexwert ( $r = 1$ ) aufnimmt.

$$z.y. \rightarrow z.y.x_r$$

- Vermehren durch Aufteilen

Eine Dimensionsvermehrung durch Aufteilen der letzten Dimension. Welche Länge die letzte (neue) Dimension haben soll, wird angegeben ( $l_x$ ).

$$z.y_j \rightarrow z.y_j x_i$$

wobei gilt:

$$l_y = l_y / l_x$$

$$h_j = (y_j - 1) / l_y$$

$$y_j = y_j - h_j * l_y$$

$$x_i = h_j + 1$$

Es ist auch möglich, daß eine bestehende Dimension in ungleiche Teile aufgeteilt wird.

- Vermindern durch Wegnehmen

$$z.y.x_1 \rightarrow z.y.$$

Alle Tupel, bei denen der Index der letzten Dimension nicht eine bestimmte Ausprägung hat, entfallen.

- Vermindern durch Vereinigen

Aus den beiden letzten Indizes wird mit Hilfe einer Linearisierungsformel der nunmehr letzte Index berechnet. Der 'alte' letzte Index entfällt.

$$z.y_j x_1 \rightarrow z.y_j$$

- Umordnen (Recorder)

$$z.y_j x_1 \rightarrow z.y_j$$

Die Reihenfolge der Indizes wird geändert.

Die oben aufgeführten Struktur- und Darstellungsänderungen werden in IAS durch einige Module realisiert. Sie enthalten im wesentlichen Routinen zur Linearisierung und zur Überführung in Tupel-Form (TUP), eine Routine zur Transponierung einer k-dimensionalen Datenmatrix in place (CHANGE) und einen Interpreter für bedingte und unbedingte Arithmetische und Boolesche Anweisungen (CONTRA). Mit Hilfe dieser Moduln ist es möglich, die von SCHAUMANN (1975) vorgeschlagene Schnittstelle  $S_0$  in wenigen Dialogschritten aufzubauen.

#### 4.2.3 Trennung zwischen Datenstruktur und Speicherstruktur und ihre Realisierung durch Programme

Wie wir oben versucht haben zu verdeutlichen, ist eines der wesentlichen Datenbankmerkmale die Trennung von Datenstruktur und Speicherstruktur und das damit verbundene Problem der Abbildung von logischen Sätzen auf physikalische. Wir haben im Rahmen von IAS durch Programme diese Abbildungen realisiert (vgl. Anhang:DZP).

Durch dieses Dateiverwaltungssystem wird völlige hardware-Unabhängigkeit erreicht. DZP wird seinerseits entweder in ein bestehendes File-Definitionsprogramm eingebettet (z.B. DEFINE FILE oder CALL DEFINE), oder es kann direkt auf absolut adressierbaren Speichermedien laufen (z.B. READDP).

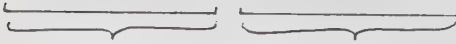
Bei der Abbildung logischer Sätze auf einen linearen "unendlichen" physikalischen Adressraum gibt es verschiedene Möglichkeiten. Dabei interessiert zum einen die Relation von logischem Satz zu physikalischem Satz, zum anderen die Zugriffsform (reiner Randomzugriff oder sequentieller Zugriff).

##### a) Realisierung bei festgelegter Relation zwischen logischem und physikalischem Satz

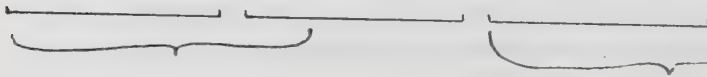
- (1) logischer Satz ist kleiner als physikalischer (ganzzahlige Anzahl von logischen Sätzen pro physikalischem Satz).



(2) beide sind gleich groß



(3) logischer Satz ist größer als physikalischer  
(eine ganzzahlige Anzahl von physikalischen Sätzen  
wird von einem logischen Satz verbraucht)



b) Realisierung bei nicht festgelegter Relation zwischen  
logischem und physikalischem Satz

Ein logischer Satz kann größer, gleich oder kleiner als  
ein physikalischer Satz sein. Dabei müssen weder eine  
ganzzahlige Anzahl von logischen Sätzen in einem physi-  
kalischen untergebracht werden, noch umgekehrt muß eine  
ganzzahlige Anzahl von physikalischen Sätzen von einem  
logischen Satz verbraucht werden.



Für eine Pufferorganisation ist allerdings wichtig, ob

- (1) ein reiner random-Zugriff erfolgt oder
- (2) ein sequentieller Zugriff auf alle Sätze erfolgen soll (in diesem Fall können OPEN- und CLOSE-Routinen eingeführt werden und dadurch die Zeit optimiert werden).

Im Anhang sind unter der Bezeichnung DZP (Dateizugriffsprogramm) alle oben aufgeführten Möglichkeiten in FORTRAN programmiert wiedergegeben.

#### 4.3 Retrieval-System RS-II

Von einem dritten vorhandenen System (RS-II) zur ernährungswissenschaftlichen Literaturdokumentation mit kleineren EDV-Anlagen (FRIEDRICH, 1976) seien hier nur die Weiterentwicklungen bezüglich der Invertierung, sowie die Möglichkeiten zur Realisierung von Mengenoperationen behandelt.

##### 4.3.1 Entwicklung einer Methode zur Invertierung grösserer Datenbestände auf kleineren EDV-Anlagen

Bei der Sekundärindizierung (Invertierung) steht jeweils hinter einer Deskriptoren-Nummer die Nummer der Dokumente, in denen dieser Deskriptor vorkommt. Die Anzahl der Dokumenten-Nummern, die hinter einem Deskriptor zu stehen kommen, ist unvorhersehbar und schwankt stark.

Für alle Deskriptoren eine Maximalanzahl von Plätzen vorzusehen, ist nicht praktikabel. Ebenso erweist sich die Möglichkeit, eine Invertierung über eine Binärmatrix zu erreichen, als undurchführbar: Zum einen ist in einem Dokumentationssystem das Produkt aus Dokumenten und Deskriptoren zu groß, als daß es in einer Binärmatrix auf kleineren Rechnern untergebracht werden könnte; zum anderen ergibt sich ein äußerst zeitintensives Updating-Verfahren, da entweder für jeden Deskriptor ein Zugriff notwendig ist (bei einer Speicherungsform Deskriptoren mal Dokumente) oder (bei einer Speicherungsform Dokumente mal Deskriptoren) ein ebenso aufwendiger Transponierungsvorgang notwendig wird (dieser muß außerdem wegen des begrenzten Speicherraums in place (in situ) durchgeführt werden).

Für die Totalinvertierung einer größeren Datenbank entwickelten wir daher ein eigenes Verfahren, das hier beschrieben werden soll.

Ausgangspunkt ist die vorliegende Eingabeform: Dokumenten-Nummer gefolgt von einer variablen Anzahl von Deskriptoren-Nummern.

| <u>Dok-Nr.</u> | <u>Deskriptoren</u> |     |     |     |
|----------------|---------------------|-----|-----|-----|
| 1              | 34                  | 74  | 111 |     |
| 2              | 74                  | 111 | 134 | 275 |
| 3              | 111                 | 234 |     |     |

Für jedes Dokument werden soviel Tupel gebildet wie Deskriptoren vorhanden sind. Alle Tupel aller Dokumente werden hintereinander gestellt.

(34,1), (74,1), (111,1) ; (74,2), (111,2), (134,2), (275,2);  
(111,3), (234,3)

In der von uns realisierten Organisationsform wird jeweils ein Element eines Tupels auf einer Speichereinheit abgesetzt.

|    |    |    |     |    |     |     |     |     |     |
|----|----|----|-----|----|-----|-----|-----|-----|-----|
| 1. | 34 | 74 | 111 | 74 | 111 | 134 | 275 | 111 | 234 |
| 2. | 1  | 1  | 1   | 2  | 2   | 2   | 2   | 3   | 3   |

Durch einen unten näher beschriebenen Sortieralgorithmus wird nach Deskriptoren sortiert:

|      |    |    |    |     |     |     |     |     |     |
|------|----|----|----|-----|-----|-----|-----|-----|-----|
| 1.   | 34 | 74 | 74 | 111 | 111 | 111 | 134 | 234 | 275 |
| 2.   | 1  | 1  | 2  | 1   | 2   | 3   | 2   | 3   | 2   |
| Pos. | 1  | 2  | 3  | 4   | 5   | 6   | 7   | 8   | 9   |

Danach wird eine Verweisliste für die Deskriptoren aufgebaut.

| <u>Deskriptor</u> | <u>Anfangsposition</u> |
|-------------------|------------------------|
| 34                | 1                      |
| 74                | 2                      |
| 111               | 4                      |
| 134               | 7                      |
| 234               | 8                      |
| 275               | 9                      |
| -                 | 10                     |

Aus diesem Index wird also auf die variabel lange Dokumentennummern-Liste ge-pointet. Nunmehr kann der Inhalt des Speichermediums entfallen. Dieses Verfahren gehört zu den im Kapitel über das Relationalmodell beschriebenen Normalisierungsverfahren.

Der von uns entwickelte Sortieralgorithmus mußte eine sehr große Datenmenge sortieren können. Es zeigte sich, daß dieses Problem nur durch Kombination eines internen mit einem externen Sortierverfahren zu lösen war.

Dabei wird jeweils ein Block, der aus zwei Hälften zusammengesetzt war, intern durch ein von uns modifiziertes QUICK-Sort-Verfahren sortiert. Die Zusammensetzung der Blockhälften wird nach dem BUBBLE-Sort-Prinzip als externes Verfahren durchgeführt.

Die folgende Abbildung zeigt einen File mit verschiedenen 'Teil'-Blöcken der Länge 1:

|   |   |   |   |   |
|---|---|---|---|---|
| A | B | C | D | E |
|---|---|---|---|---|

Ein Block ist wahlfrei zugreifbar. Zwei Teilblöcke, die jeweils zusammengefaßt werden, sind nur so groß, daß sie intern sortierbar sind (d.h. sie überschreiten nicht die Arbeitsspeicherkapazität).

Es werden nun Teilblöcke A und B gelesen. Sie bilden einen Vektor der Länge  $2k$ , dieser wird QUICK-sortiert und anschließend wieder in zwei Teilblöcke der Länge  $k$  zerlegt. Der zweite Teilblock wird auf die ursprüngliche Position B zurückgeschrieben (alle Elemente in A sind nun kleiner als die in B). Es wird C gelesen, mit A zusammen ein Vektor gebildet, dieser sortiert, danach geteilt, der zweite Teil nach C zurückgeschrieben, usf.

Nach dem Zurückschreiben von E wird A zurückgeschrieben. In A befinden sich nun die kleinsten  $k$  Elemente des gesamten Files. Der Vorgang wird mit B und C uws. fortgesetzt.

Da beim QUICK-Sort-Verfahren eine extreme Beschleunigung dadurch erreicht wird, daß Teilstrings bereits sortiert vorhanden sind, fallen die internen Sortierzeiten kaum ins Gewicht. Wir haben das Verfahren als "Meta-Bubble-QUICK" (MBQ)-Verfahren bezeichnet.

#### 4.3.2 Möglichkeiten zur Realisierung von Mengenoperationen

Sind, wie oben beschrieben, auf einem Medium für alle vorhandenen Deskriptoren jeweils alle Dokumenten-Nummern abgespeichert, so lassen sich für jeweils zwei Deskriptoren Vektoren A und B aufbauen (über Stream-I/O-Verfahren), die die dazugehörigen Dokumentenzahlen enthalten.

Die Operatoren AND, OR, XOR (exklusives Oder) und ANDNOT lassen sich durch Merge-Vorgänge realisieren. Die Ergebnismenge der Operation wird in einem dritten Vektor C abgelegt (vgl. im Anhang: UP SETOP). Dabei kann der ungünstigste Fall eintreten, daß  $C = A + B$  gleich der Anzahl aller vorhandenen Dokumente ist, womit dieses Verfahren nicht mehr im Arbeitsspeicher durchzuführen ist.

In diesem Fall werden die aus den Vektoren entnommenen Dokumenten-Nummern als Ordinalzahlen für Positionen in einem Bitstring aufgefaßt. Die betreffende laufende Bit-Position wird auf "1" gesetzt. Zur Realisierung der Mengenoperatoren genügt dann das einfache Verschieben einer Maske über zwei Bitstrings.

### III Diskussion und Zusammenfassung

#### 1. Diskussion

Ziel der vorliegenden Arbeit war es, die Anwendbarkeit von Datenbankorganisationsmethoden zu untersuchen. Wir wählten dabei zwei Ansätze.

Unter theoretischen Aspekten standen zur Prüfung der Anwendbarkeit von Datenbankorganisationsprinzipien eine ausführliche Darstellung der Methoden der Datenorganisation und eine Analyse umfassender Datenbankmodelle im Mittelpunkt.

Unter mehr praktischen Aspekten versuchten wir, den Nachweis der Anwendbarkeit dadurch zu erbringen, daß wir für verschiedene Fragestellungen Datenorganisationsprinzipien praktisch realisierten.

Intention des theoretischen Ansatzes war es, die Struktur und Komplexität der Prinzipien zu erläutern, um damit einen Einblick in die implizit gegebene Utilität und Transportierbarkeit dieser Ideen, Algorithmen, Verfahren und Modelle zu ermöglichen. Dies sei an drei Punkten erläutert.

- a) Die Gegenüberstellung der mittleren Zugriffszahlen der Suchverfahren ohne Adressrechnung ist ein wesentlicher Gesichtspunkt für die praktische Wahl eines Verfahrens, der allerdings nicht ohne die "Update"-Möglichkeiten gesehen werden darf. Erst die gemeinsame Bewertung beider Kriterien, gemessen an den in der Realität bestehenden Forderungen, kann zu einer Auswahl einer bestimmten Methode führen.

- b) Bei der Beurteilung eines Baumverfahrens (Speicherung von nicht-linearen Listen) genügt sicher nicht die Bewertung der Komplexität (oder Einfachheit) eines Algorithmus. Als wesentliches Kriterium wird die Kontrolle des Wachstums eines Baumes herangezogen werden müssen. Wird dies gewährleistet, ist weiterhin zu analysieren, wie die Schwierigkeit der notwendigen variablen Pointeranzahl pro Knoten überwunden wird. Ferner wird von großem Interesse sein, mit welchem Aufwand das Löschen von Knoten realisiert werden kann. Schließlich wird man zu einer Beurteilung auch eine Abschätzung der Möglichkeiten und Schwierigkeiten der Gesamt-Reorganisation eines baumstrukturierten Datenbestandes mit heranziehen.
- c) Die in diesem Zusammenhang komprimierte Darstellung von umfassenden Datenbankmodellen legt durch ihre Deskription u.E. folgende Schlußfolgerungen nahe:
- Es ist (zumindest zum jetzigen Zeitpunkt noch) ausgeschlossen, derart komplexe Modelle auf kleineren Rechnern zu implementieren.
  - Zweck der Darstellung war demnach auch, durch die Beschreibung der Grundmodelle diejenigen Grundprinzipien herauszuheben, die sich für eine Anwendung in kleinerem Rahmen eignen. Dies sind z.B. beim CODASYL-Modell die Unterscheidung zwischen Schema und Unterschema, was in die Richtung einer Schichtenarchitektur einer Datenbank weist, oder das "current unit"-Konzept, das sich besonders für die Bewältigung des Logging-Problems als geeignet erweist (Wiederanlaufverfahren bei Systemzusammenbrüchen).
- Beim Relationalmodell geht aus der beschreibenden Darstellung hervor, daß sich der Vorgang der Normalisierung, sowie einige Operationen mit Tupeln relativ leicht realisieren lassen.

Die Beschreibung von Datenbankorganisationsprinzipien stellt somit die notwendige Voraussetzung für die Untersuchung der Anwendbarkeit und erst recht für die Anwendung (Realisierung) dar.

Die zu Beginn des zweiten Teils der Arbeit von uns formulierten Thesen sind als Leitlinien oder Grundsätze für das konkrete System-Design aufzufassen. Es sind Idealforderungen. Man kann sie wie folgt zusammenfassen:

Ein System soll modular (nach dem Baukastenprinzip (Tool-Pool)) aufgebaut sein (These 1), die einzelnen Teile sollen dabei unabhängig voneinander sein. Die Verwirklichung dieses formalen Prinzips ist im System REMED zu verfolgen. Hier unterscheiden wir Recherchemodul A und B (Vorauswahl über invertierte Datei und eigentliche Suche), die ihrerseits wiederum aus mehreren (allgemeinen) Unterprogrammen bestehen.

Die Forderung nach dem schrittweisen Aufbau eines Systems (These 2) bedingt die Existenz eines Gesamtkonzepts, das sich strukturell nicht oder nur sehr geringfügig ändert. Die verallgemeinerte Forderung wäre die Forderung nach einem "offenen" System. Genau diese Charakteristik weist das System IAS auf. Es ist als offenes System konzipiert und wird ständig durch weitere statistische Auswerteverfahren erweitert. Da grundsätzlich die Datenstrukturen an die Programme angepaßt sind, sind im Prinzip alle bestehenden Statistikprogramme als Bausteine für IAS verwendbar.

Die aufgestellte Forderung nach einem strukturierten Entwurf (These 3) haben wir im System RS-II erfüllt. Es wurde im Top-down-design mit der Methode des step-wise refinement entworfen und ausgeführt. (In den 1 1/2 Jahren seines Routine-Gebrauchs ist kein Fehler aufgetreten.)

Die Forderung nach einem Team, das nach einer Checkliste ein System-Design ausführt (These 4), konnte bisher nur in Ansätzen verwirklicht werden (RS-II). Wir halten aber die Orientierung am potentiellen Benutzer für eine äußerst wichtige Forderung. Das Innovationsproblem muß gelöst werden, wenn Systeme vom Benutzer nicht abgelehnt werden sollen.

Die von uns aufgestellten Grundsätze sind, wie gezeigt wurde, in den drei erwähnten Systemen schwerpunktmäßig erfüllt. Nicht durchgängig erfüllt ist insbesondere die Forderung nach einem strukturierten Design ("Maß der Strukturiertheit als Funktion der Entstehungszeit").

Gehen wir in unserer Diskussion weiter zum Aspekt der "Arbeitsmittel" (Beschreibungs- und Programmiersprache), so ist als ein Punkt festzuhalten, daß eine allgemein strukturierte Notierung verwendet werden sollte. Dies ist nur ein Sektor der gesamten Konstruktionsproblematik von Software. Die Erörterung sollte im Rahmen der modernen Design-Methoden und -Hilfsmittel (improved programming techniques) gesehen werden. Gleichzeitig ist eine Relativierung der strengen strukturierten Programmierung notwendig. Es scheint uns angemessener, eine "disziplinierte Programmierung" zu fordern (vgl. den programmatischen Titel einer Monographie von DIJKSTRA (1975): A discipline of programming).

Um, wie oben bereits erwähnt, die in Programmsystemen verwirklichten Prinzipien nutzbringend anzuwenden, muß eine Kommunikation (Interaktion) mit dem Benutzer möglich sein. Diese Interaktionsmöglichkeit soll komfortabel sein. Die im Rahmen dieser Arbeit entwickelten I/O-Moduln sind für allgemeine Anwendungen konzipiert, mit ihnen sollen im Verbund mit jeweils spezifisch ausgewählten Datenbankmoduln benutzernahe Systeme erstellt werden.

Das Konzept des Formatgenerators ist in verschiedenen Formen realisiert worden, so z.B. für die On-line-Eingabe und -Korrektur von statistischen Untersuchungsdaten (IAS).

Ebenso existiert der Formelinterpreter für eine Abfragesprache in mehreren Versionen. In REMED wird eine Fassung verwendet, die FORTRAN-Anweisungen interpretiert, während in RS-II ein Formelinterpreter enthalten ist, der eine Übergangsmatrix zugrundelegt und (erweiterte) boolesche Ausdrücke mit besonderen Operanden interpretiert. Formatgenerator und Formelinterpreter können verschiedenen Zwecken leicht angepaßt werden.

Im folgenden sei abschließend auf einige Funktionen eingegangen, die im Rahmen der Arbeit exemplarisch veranschaulicht wurden.

Im System REMED wird eine spezielle Form der Verkettung innerhalb eines logischen Satzes verwendet. Die Sätze des Datenbestandes selbst sind nach dem Prinzip der einfachen Vorwärtsverkettung miteinander verbunden. Das Prinzip der Teilinvertierung findet in dem Modul für die Vorauswahl Verwendung. Ein Formelinterpreter für FORTRAN-Ausdrücke ermöglichte die Implementierung einer Abfragesprache.

Im Routine-Betrieb hat sich die Form der Datenstrukturierung als störanfällig erwiesen. Sie wird daher einer Revision unterzogen werden müssen. Es ist daran gedacht, den gesamten Datenbestand über eine Verweisdatei zu kontrollieren. Durch die Einführung von "current units" soll das Problem des Logging bei Systemzusammenbruch gelöst werden. Vorauswahl (über Bitstring) und eigentliche (se-

quentielle) Suche werden allerdings weiterhin getrennt bleiben, da sich dieses Prinzip bewährt hat. Vom Suchmodul wird in der weiteren Entwicklung der Ausgabeteil vollständig abgekoppelt werden.

Im System IAS wird das Prinzip der Datenunabhängigkeit konsequent angewendet. (Die Weiterentwicklung des Dateikonzeptes wird zu einem kompatiblen File-Verwaltungssystem führen.)

Das Problem der Dateneingabe mit den damit verbundenen Korrekturschleifen ist durch einen modifizierten "Formatgenerator" gelöst worden. Eingabe und Korrektur sind ein Vorgang geworden. Untersuchungen über die Struktur von Daten für statistische Auswertungen führten zu einem einzigen Datenstrukturmodell, von dem es lediglich verschiedene Darstellungsformen gibt. Die Darstellungsform wird wiederum logisch von der Speicherform getrennt. Mit Hilfe der aufgrund dieser Prinzipien entwickelten Programme ist es möglich, jede denkbare Darstellungsform zu erstellen. In der praktischen Anwendung hat sich unser IAS bewährt und ist inzwischen auf einer IBM 370 installiert.

Mit dem Retrieval-System für die ernährungswissenschaftliche Literaturdokumentation (RS-II) wird ein praktisches Beispiel für das Prinzip der Total-Invertierung von Datenbeständen gegeben. Es wird gezeigt, daß die Speicherung nach dem Prinzip der Binärmatrix nicht möglich ist.

Wie einfache Operationen mit Mengen (Relationen) in einer Programmiersprache bewältigt werden können, wird anhand der Operationen Vereinigung und Durchschnitt gezeigt.

An Systemen, die eine nicht-lineare Listenverarbeitung beinhalten (Verwendung des B-Baumprinzips zum Aufbau eines Lexikons) wird derzeit gearbeitet.

### Ausblick

Wie schon oben verschiedentlich erwähnt, halten wir die Erstellung eines Tool-Pools für eine geeignete Vorgehensweise.

Als feste Bestandteile dieses Pools sind anzusehen:

Formatgenerator (Eingabe), Formelinterpreter (Abfrage), eine portable (kompatible) Fileverwaltung, ein Modul zur Generierung von Bit-Strings (Teilinvertierung) und ein Modul zur Totalinvertierung.

Ferner wären systematisch Logging-Probleme anzugehen und Datenschutzfragen zu behandeln.

## 2. Zusammenfassung

Die vorliegende Arbeit gliedert sich in zwei Teile.

Im ersten Teil werden die Datenbankorganisationsprinzipien zusammengestellt und erläutert. Zunächst wird auf die Forderungen einer modernen Datenbankorganisation (u. a. Datenunabhängigkeit, Unterscheidung von Datenstruktur und Speicherstruktur) eingegangen. Es folgen eine Beschreibung der Such- und Änderungsverfahren mit linearen Listen und mit Bäumen, sie lassen sich, da sie gut algorithmisierbar sind, in einer kompatiblen Programm-sprache realisieren.

Am CODASYL-Modell werden Mängel aufgezeigt, die die Darstellung von Strukturbeziehungen betreffen. Es erweist sich als ein schwerfälliges und aufwendiges Instrument und wenig geeignet für den Einsatz auf kleineren EDV-Anlagen.

Prinzipien des Relationalmodells hingegen weisen sehr gute Realisierungsmöglichkeiten, z.B. in FORTRAN, auf, wie dies z.B. anhand des "join-operators" gezeigt wird.

Im zweiten Teil der Arbeit werden zunächst thesenhaft Forderungen für den Aufbau von "angemessenen" problembezogenen Datenbanksystemen gestellt.

Es folgt eine Zusammenstellung wichtiger Aspekte bei der Auswahl der Arbeitsmittel für eine Realisierung (Darstellungs- und Realisierungssprachen).

Im Bereich der Interaktion mit einer Datenbank wurde aufgezeigt, daß auch hier eine Realisierung (in FORTRAN) möglich ist. Es wird das Prinzip der Steuerung der Dateneingabe in eine Datenbank mit Hilfe eines sogenannten Formatgenerators beschrieben, und es wird gezeigt, daß als Abfragesprache eine interpretierte Programmiersprache (hier FORTRAN) herangezogen werden kann.

Danach werden drei Routine-Systeme und deren geplante Weiterentwicklungen vorgestellt, mit der Intention, einige wichtige Funktionen exemplarisch zu veranschaulichen.

Beim System REMED (REcherche Moduln zur Erschließung medizinischer Daten) wird auf die verkettete Datenstruktur und auf die Realisierung einer Abfragesprache als Interpreter eingegangen.

Im System IAS (Interaktives Auswertungs-System) wird primär auf die Trennung zwischen logischen und physikalischen Sätzen Wert gelegt, und es werden eingehend die Struktureigenschaften von statistischem Datenmaterial behandelt. Es wird dabei ein Datenstrukturmodell vorgestellt, das durch Darstellungsänderung und Strukturänderung modifiziert werden kann und das sich vom Benutzer leicht handhaben läßt. Die Eingabe von Daten wird mit Hilfe eines Formatgenerators ermöglicht.

Aus dem System RS-II (Retrieval System (Version II) für die ernährungswissenschaftliche Literaturdokumentation) und seinen Weiterentwicklungen wird eine Methode zur Invertierung großer Datenmengen vorgestellt, sowie die Möglichkeit, Mengenoperationen durch FORTRAN-Programme zu realisieren.

Für weitere Anwendungen scheint sich u.E. der Weg der Erstellung eines Datenbank-Tool-Pools von standardisierten Programmen als tragfähig zu erweisen. Er stellt eine brauchbare Möglichkeit dar, um zu "angemessenen" Lösungen von spezifischen Problemen im Bereich medizinischer Dokumentation und Statistik zu kommen.

Literaturverzeichnis

- ACHESON, E.D. Medical Record Linkage  
Methods of Information in Medicine 8 (1969) S. 1-6
- ADELSON-VELSKII, G.M.; Ein Algorithmus zur Informationsorganisation (orig.russ.,  
LANDIS, E.M. engl.Übers. in:) Soviet Maths.  
3, s. 1259-1263 zitiert nach  
WIRTH, 1975
- AHRENS, F.; WALTER, H. Datenbanksysteme  
Berlin, N.Y.: DeGruyter, 1971
- ASTRAHAN, M.M. et al. System R. A relational Approach  
to Data Base Management  
IBM-Research Paper, RJ 1738,  
San Jose, 1976 in: Lehrgangsun-  
terlagen Sommerseminar über Da-  
tenbanktechnologie 12.-16.7.1976,  
GMD-Birlinghoven
- BABAD, J.M. A Record and File Partitioning  
Model Communications of the ACM  
20 (1977) 1, s. 22-31
- BAGINSKI, P. Eine Rechenmethode für zeitkri-  
tische, merkmalsorientierte Such-  
prozesse bei Datenbanken  
adl-Nachrichten, 64(1970)S.392-395
- BAYER, R.; MCCREIGHT, E. Organisation and Maintenance of  
Large Ordered Indexes;Acta Infor-  
mation, 1 (1972) 3, S. 173-189

- BERZTISS, A.T.                      Data Structures, Theory and practice  
London, N.Y.: Academic Press, 1975
- CARDENAS, A.F.                      Evaluation and Selection of File  
Organisation, A Model and System  
Communication of the ACM, 16 (1973)  
S. 540-548
- CARDENAS, A.F.                      Analysis and Performance of Inverted  
Data Base Structures  
Communications of the ACM, 18 (1975)  
5, S. 253-263
- CHAPIN, N.                          Common file organisation techniques  
compared  
in: AFIPS Conference Proceedings Vol. 35  
1969, S. 413-422
- CODASYL                              CODASYL DATA BASE TASK GROUP REPORT  
DTBG-Report  
Bezug: IFIP Administrative Data  
Processing Group 40 Paulus Potter-  
straat, Amsterdam
- CODD, E.F.                          A relational model for large shared  
data banks  
Communications of the ACM, 13(1970)  
6, S. 377-387
- COOPER, B.E.                      The Continuing Developments of a  
Statistical System  
In: MILTON, R.C.; NELDER, J.A.  
(Hrsg.) Statistical Computation  
London, N.Y.: Academic Press, 1969

- CZEGLEDY, P.F.; LINDER, A. User's manual of EDSGN, Genf: Laboratoire de Statistique mathématique, 1975
- DAHL, O.J.; DIJKSTRA, E.W. Structured Programming  
HOARE, C.A.R. N.Y., London: Academic Press, 1972
- DUDECK, J. et al. Erste Erfahrungen mit dem Labor-Computersystem CLINICAL LAB 12  
Medizinische Technik, 5(1972), S. 189-198
- DIJKSTRA, E.W. GOTO Statements considered harmful  
Communications of the ACM, 11 (1968) S. 147-148
- DWORATSCHECK, S. Management Informations-Systeme  
Berlin, N.Y.: De Gruyter, 1971
- ELSON, M. Data Structures  
Stuttgart: Science Research Associates Inc., 1975
- FRIEDRICH, H.-J. FORTRAN-Standards, Empfehlungen für die standardisierte Anwendung von FORTRAN für Programme zur Auswertung medizinischer Datenbestände  
Beiheft B1 zum Statistical Software Newsletter, Februar, 1975
- FRIEDRICH, H.-J. Datenvorbereitung als Datenstrukturänderung, Speicherstrukturänderung und Datenmodifikation (Vorschläge zur Realisierung in interaktiven Programmsystemen)  
Statistical Software Newsletter 2(1976) 3, S. 68-71

- FRIEDRICH, H.-J. Ernährungswissenschaftliche Literaturdokumentation auf kleineren EDV-Anlagen  
Ernährungs-Umschau, 12(1976), S. 371-373
- FUCHS-KITTOWSKI, K. Informatik und Automatisierung  
et al. Band I: Theorie und Praxis der Struktur und Organisation der Informationsverarbeitung  
Berlin (O): Akademie Verlag, 1976
- GIERE, W.; HEGER, J.; Richtlinien für die Formulierung  
KRIER, N. von Strukturblocken in FORTRAN  
Hektogr. Manuskript, S. 9, DKD, 1976
- HARTMAN, W.; MATTHES, H. Management information systems  
PROEME, A. handbook, New York, 1968
- HASSELMEIER, H.; Data Base Systems  
SPRUTH, W.G. (Hrsg.) Proceedings of the 5<sup>th</sup> Informatik  
Symposium IBM-Germany 24.-25.9.1975  
= Bd 39 Lecture Notes in Computer  
Science, Hrsg.: GOOS, HARTMANIS  
Berlin, Heidelberg, N.Y.: Springer 1976
- HELD, G.; STONEBREAKER, M. Storage Structures and Access Methods  
in the Relational Data Base Management System INGRES  
ACM-Conference on Data, its use, organisation and Management  
San Francisco, April 1975
- HENSELMANN, G. Datenstrukturen in einem Datenbank-  
System für Minicomputer  
Vortrag auf der Frühjahrstagung der  
GMDS-GI vom 30.3.-20.4.1977, Gießen

- HSIAO, D.; HARARY, F.      A Formal System for Information  
Retrieval from Files  
Communication of the ACM, 13 (1970),  
S. 67-73
- Infotech State of the Art Report DATA BASE SYSTEMS  
Maidenhead:Infotech Information,1975
- JANKO, W.                      Zur optimalen Blocklänge beim m-  
Wege-Suchverfahren  
Angewandte Informatik,11(1976)S.487-489
- JARDINE, D.A. (Hrsg.)      Data Base Management System  
Proceedings of the SHARE Working  
Conference Montreal, 23.-27.7.1973  
London, N.Y.: North Holland Publishing  
Comp., 1974
- KNUTH, D.E.                      The Art of Computer Programming  
Vol. 1 Fundamental Algorithms  
Reading (Mass.):Addison-Wesley,1968
- KNUTH, D.E.                      The Art of Computer Programming  
Vol. 3 Sorting and Searching  
Reading: Addison-Wesley, 1973
- LEFKOWITZ, P.D.                  File Structures for On-line-Systems  
N.Y.:Spartan Books, 1969
- LIMBURG, L.A.H.;  
MACHMER, L.                      Tabellenverarbeitung  
Zeitschrift für Datenverarbeitung  
8(1970) 8, S. 531-540
- LIU, J.-H.                      Application of the Relational Model  
to Record Indexing and Retrieval  
Computers and Biomedical Research  
10(1977), S. 25-34

- LUM, V.Y. Multi-attribute Retrieval with Combined Indexes  
Communications of the ACM, 13(1970)  
11, S. 660-665
- LUM, V.Y.; YUEN, P.S.; DODD, M. Key to Address Transform Techniques,  
a fundamental performance Study on  
large Existing Formated Files  
Communications of the ACM, 14(1971),  
S. 228-239
- LUTZ, T. Das computerorientierte Informations-  
System (CIS)  
Berlin, N.Y.: De Gruyter, 1973
- LUTZ, T. Datenbanken  
Stuttgart: Science Research Associates  
GmbH, 1976
- LUTZ, T.; KLIMESCH, H. Die Datenbank im Informationssystem  
München, Wien: Oldenbourg, 1971
- MAURER, W.D.; LEWIS, T.G. HASH Table Methods  
ACM Computing Surveys, 7(1975), S.5-19
- MERTEN, H. Datenbankorganisation  
Köln, Braunsfeld: Verlagsges. R.Müller,  
1974
- MORRIS, R. Scatter Storage Techniques  
Communications of the ACM, 11(1968)1,  
S. 38-43
- MÜLLER, R. Ein Informationssystem mit ADABAS  
Computerpraxis 11(1970) S. 210-215

- NIEMEYER, G.                      Dateiorganisation und -verarbeitung  
München: Verlag Vahlen, 1975
- NELDER, J.A.                      The Description of Statistical Data  
Structures for Statistical Computing  
in: MILTON, R.C.; NELDER, J.A.(Hrsg.)  
Statistical Computation London,N.Y.:  
Academic Press, 1969
- NOLTEMEYER, H.                      Datenstrukturen und höhere Program-  
miertechniken  
Berlin, N.Y.: De Gruyter, 1972  
= GÖSCHEN 5012
- RUSTIN, R. (Hrsg.)                      Data Base Systems  
Courant Computer Science Symposium 6  
24.-25.5.1971, Englewood Cliffs:  
Prentice Hall, 1972
- SAGER, W.                          Konzept für ein Datenerfassungssystem  
formatierter Daten  
Systemvorgabe eines Formatgenerators  
für eine Herstellerfirma (Hektogra-  
fiertes Script, 19 S.) Gießen: Inst.  
f. Med. Stat. u. Dok., 1976
- SALTON, G.                          Automatic information organisation  
and retrieval  
N.Y.: McGraw Hill, 1968
- SCHAUMANN, P.                      Eine allgemeine standardisierte Daten-  
schnittstelle zur Verbindung von Da-  
tenaufbereitungs- und Auswertungstei-  
len bei statistischen Programmen  
Statistical Software Newsletter  
1(1975), S. 29-31

- SCHROEDER, K.                    Vergleich der Verweistechiken in  
Datenbanksystemen: Adresskettung  
contra Indextabellen  
Angewandte Informatik 4(1972),  
S. 145-153
- SELBMANN, H.K.                Bit-String Processing for Statistical  
Evaluation of Large Volumes of Medical  
Data  
Methods of Information in Medicine  
13(1974) 2, S. 61-64
- SEIDEL, J.                    Eine teilweise Implementierung des  
DBTG-Vorschlages für Unterrichts-  
zwecke (DAMOS)  
Birlinghoven: GMD, 1975
- SIX, H.W.                    Verbesserung des m-Wege-Suchverfahrens  
Angewandte Informatik 2(1973), S.79-83
- TODD, S.                    Integrated Architecture for transac-  
tion specification and optimization  
in relational Data-Base Systems  
Preprint, IBM-UK Scientific Center  
Peterlee, 1976  
in: Lehrgangsunterlagen Sommersemi-  
nar über Datenbanktechnologie  
vom 12.-16.7.1976, EMD-Birlinghoven
- WANG, C.P.; WEDEKIND, H.      An approach for segment synthesis  
in logical data base design  
IBM-Research-Report RJ 1397  
San Jose, 1974
- WEDEKIND, H.                Datenorganisation  
Berlin: DeGruyter, 1970, 1972 2. Aufl.

WEDEKIND, H.

Datenbanksysteme I  
Mannheim, Wien, Zürich: B-IWissen-  
schaftsverlag, 1974  
= Reihe Informatik 16

WEDEKIND, H.

Der Entwurf von Datenbanksystemen  
nach dem Relationalmodell  
in: Informatik Anwendungen, Bei-  
träge zum Kolloquium am 3. und 4.  
Febr. 1975 im wiss. Zentrum für In-  
formatik, Hrsg.: Präsident der TH  
Darmstadt, S. 11-33

WEDEKIND, H.; HÄRDER, T.

Datenbanksysteme II  
Mannheim, Wien, Zürich: B-I Wissen-  
schaftsverlag, 1976  
= Reihe Informatik 18

WIRTH, N.

Algorithmen und Datenstrukturen  
Stuttgart: Teubner, 1975  
= Teubner Studienbücher Informatik 31

Anhang

- Zu II 3.1 Eingabe von Daten in die Datenbank  
Formatgenerator DEP
- Zu II 3.2 Abfrage von Daten  
Formelinterpreter CONTRA
- Zu II 4.1 Recherchemoduln zur Erschließung medizinischer Daten  
Programmsystem REMED
- Zu II 4.2 Interaktives Auswertungs-System für medizinisch-statistische Daten
- 4.2.2 Datenstrukturen von medizinisch-technischen Daten, Realisierung von Struktur- und Darstellungsänderungen durch Programme  
Programmpaket TUP, Unterprogramm CHANGE
- 4.2.3 Trennung zwischen Datenstruktur und Speicherstruktur und ihre Realisierung durch Programme  
Programmpaket DZP
- Zu II 4.3 Retrievalsystem zur ernährungswissenschaftlichen Literaturdokumentation
- 4.3.1 Methode zur Invertierung größerer Datenbestände auf kleineren EDV-Anlagen.  
Programm MBQ
- 4.3.2 Möglichkeit zur Realisierung von Mengenoperationen  
Unterprogramm SETOP

Aus technischen Gründen konnte der umfangreiche Anhang der gedruckten Arbeit nicht beigelegt werden.

Kopien des Anhangs sind auf Anfrage beim Verfasser erhältlich.

## Verzeichnis meiner akademischen Lehrer

Physiologie : FLECKENSTEIN (Freiburg), HENSEL (Marburg)

Methodenlehre : FAHRENBURG (Freiburg), JANZE (Gießen)

Psychologische Diagnostik : WEWETZER (Gießen)

Pädagogische Psychologie : TODT (Gießen)

Med. Statistik und Dokumentation : DUDECK (Gießen), ILM (Marburg)



### Lebenslauf

- 13.9.1943 in Aussig a.d. Elbe geboren  
Vater: Postbeamter Franz Friedrich  
Mutter: Hertha Friedrich, geb. Barth
- 6.3.1963 Reifeprüfung am neusprachlichen Gymnasium  
in Bad Iburg
- 1963 / 65 Bundeswehrzeit, Entlassung als Leutnant der  
Reserve
- SS 1965 Aufnahme des Studiums der Psychologie in Freiburg
- 18.4.1968 Vordiplom, danach Wechsel nach Gießen
- 21.12.1970 Hauptdiplomprüfung für Psychologen
- ab 1971 Wissenschaftlicher Mitarbeiter am Institut  
für medizinische Dokumentation und Statistik  
im Fachbereich Humanmedizin der Universität  
Gießen;  
Nachuniversitäre Ausbildung zum wissenschaftlichen  
Dokumentar;  
Aufbauorganisation und stellvertretender Leiter  
der Schule, bzw. der Staatlichen Lehranstalt  
für Medizinische Dokumentationsassistenten  
an der Universität Gießen;  
Lehraufträge für Medizinische Statistik im Fach-  
bereich Humanmedizin der Universität Gießen;  
wissenschaftliche Tätigkeit in der klinisch-wissen-  
schaftlichen Informationsverarbeitung
- SS 1974 Aufnahme des Ergänzungsstudiums der Humanbiologie  
an der Universität Marburg
- ab Mitte  
1977 Übernahme eines Projektes des BMFT zur Curriculum-  
entwicklung der Lehranstalten für Medizinische  
Dokumentationsassistenten
- seit 1971 verheiratet, drei Kinder.





